Information Builders®

FOCUS for S/390

DB2 and SQL/DS Read/Write Interface User's Manual
Version 7.0

DN1000048.0299

Contents

Preface

This manual describes how to use the FOCUS® DB2® and SQL/DS™ Interface. It applies to FOCUS Version 7.0 and to all DB2 and SQL/DS releases, except where noted.

With the FOCUS DB2 and SQL/DS Interface installed, you can use FOCUS to update, analyze, and report from data stored in tables and views of the DB2 or SQL/DS Relational Database Management System (RDBMS).

To use the Interface effectively, you must be familiar with basic FOCUS reporting and database maintenance procedures. You must also have access to DB2 or SQL/DS relational table information, including the actual data within the tables and views themselves, and descriptive information about them (table names, column names, and datatypes). Check with your RDBMS database administrator (DBA) about table descriptions, storage, and other site-specific considerations.

You should use this manual in conjunction with the *FOCUS for IBM® Mainframe Documentation Set Release 7.0 PLUS MAINTAIN* (DN1001005.0495), which includes the *FOCUS for IBM Mainframe User's Manual* and the *FOCUS for IBM Mainframe MAINTAIN User's Manual*.

This manual incorporates Version 7.0 new features and Technical Memo TM7922, *DB2 Interface BIND Parameter Settings*.

How This Document Is Organized

Chapter/Appendix		**Description**
1	*Introduction*	This chapter describes how the Interface functions and how to use it.
2	*Getting Started*	This chapter explains how to invoke the Interface for DB2 and SQL/DS. It provides sample CLISTs, EXECs, and JCL.
3	*Security*	This chapter discusses RDBMS and FOCUS security.
4	*Describing Tables to FOCUS*	This chapter describes FOCUS single-table Master and Access Files and explains how to use a FOCUS OCCURS segment.
5	*Multi-table Structures*	This chapter describes optional multi-table Master and Access Files.

Chapter/Appendix		**Description**
6	*Automated Procedures*	This chapter describes the AUTODB2 and AUTOSQL facilities that automatically generate FOCUS file descriptions, and the FOCUS CREATE FILE command that creates RDBMS tables.
7	*The Interface Optimizer*	This chapter discusses the Interface Optimizer and how the Interface passes certain data manipulation operations to the RDBMS for processing. It also describes how to evaluate report requests with the FOCUS EXPLAIN Utility.
8	*Advanced Reporting Techniques*	This chapter describes advanced reporting topics such as the TABLEF command, extract files and the HOLD FORMAT SQL option, the FOCUS JOIN command, short paths and the SET ALL command, and structures with multiple paths and unique relationships.
9	*Direct SQL Passthru*	This chapter describes how to issue Interface and SQL commands from the Direct SQL Passthru facility and how to include parameter markers in Direct SQL Passthru commands.
10	*Thread Control Commands*	This chapter describes commands for connecting to and disconnecting from the RDBMS, for automatically issuing COMMIT WORK commands, and for opening and closing DB2 threads.
11	*Environmental Commands*	This chapter lists Interface environmental commands that display and change the parameters that govern your FOCUS session. It also lists the Dialogue Manager variables that store the values of those parameters.
12	*Maintaining Tables With FOCUS*	This chapter describes table maintenance, RDBMS-managed integrity and concurrency, FOCUS-managed integrity and concurrency, and error handling within a maintenance procedure.
13	*Static SQL*	This chapter describes how to compile report and maintenance procedures that contain SQL statements.
14	*CALLDB2: Invoking Subroutines Containing Embedded SQL*	This chapter describes the CALLDB2 subroutine with which you can invoke user-written subroutines that contain embedded SQL calls to DB2.

Chapter/Appendix		**Description**
A	*Additional Topics*	This appendix describes problem-solving aids such as the RDBMS return code. It also discusses standard FOCUS and Interface differences, long fieldname considerations, default DATE considerations, and remote segment descriptions.
B	*Native SQL Commands*	This appendix describes how to issue native SQL commands from FOCUS and lists SQL commands supported within FOCUS.
C	*File Descriptions and Tables*	This appendix contains the Master and Access Files for the examples that are included throughout the manual.
D	*Tracing Interface Processing*	This appendix discusses the trace facilities provided by the Interface.
E	*SQL Codes and Interface Messages*	This appendix lists common SQL return codes and describes how to access Interface error codes and explanations.
F	*Additional DB2 Topics*	This appendix describes support for the DB2 Distributed Data Facility, support for DB2 Distributed Relational Database Architecture® (DRDA®), and read-only access to IMS data from DB2 MODIFY procedures.

Documentation Conventions

The following conventions apply throughout this manual.

Convention	**Description**
`THIS TYPEFACE`	Denotes a command that you must enter in uppercase, exactly as shown.
`this typeface`	Denotes a value that you must supply.
`{ }`	Indicates two choices from which you must choose one. You type one of these choices, not the braces.
`|`	Separates two mutually exclusive choices in a syntax line. You type one of these choices, not the symbol.
`[ ]`	Indicates optional parameters. None are required, but you may select one of them. Type only the information within the brackets, not the brackets.
<u>`underscore`</u>	Indicates a default setting.
`...`	Indicates that you can enter a parameter multiple times. Type only the information, not the ellipsis points.
. . .	Indicates that there are (or could be) intervening or additional commands.

New Features/Enhancements

The following chart summarizes the new features by chapter:

Chapter/Appendix		**New Features**
2	*Getting Started*	The distribution tape provides sample run-time CLISTs and JCL.
4	*Describing Tables to FOCUS*	ASC and DESC are synonyms for LOW and HIGH in the KEYORDER attribute of the Access File. PRECISION attribute in the Access File controls host variable length in Static TABLE requests. Full support for the long DECIMAL datatype.

Chapter/Appendix		**New Features**
6	*Automated Procedures*	SET IXSPACE can override the default index space parameters for indexes created by the FOCUS CREATE FILE command
7	*The Interface Optimizer*	The Interface now optimizes: • Screening conditions on DEFINE fields that derive their values from a single segment in a join structure, even if Interface join optimization is disabled. • Certain occurrences of FOCUS FST. and LST. aggregation operators. • Most joins, including many that FOCUS handled in prior releases. • Joins between heterogeneous file types. Available as of FOCUS 7.0.6. • Aggregation on DEFINE fields referenced in BY clauses. Available as of FOCUS 7.0.6. • LIKE with escape. Available as of FOCUS 7.0.9. • IF-THEN-ELSE DEFINE expressions (DB2 Interface only). Available as of FOCUS 7.0.8. There are new EXPLAIN FOCEXECs for: • DB2 Version 3. • SQL/DS Version 3 Release 4.
8	*Advanced Reporting Techniques*	Full support for the long DECIMAL datatype. SET IXSPACE can override the default index space parameters for indexes created by HOLD FORMAT DB2. Outer join optimization. Available as of FOCUS 7.0.6 Control over outer join optimization with the SET SQLJOIN OUTER command. Available as of FOCUS 7.0.9
9	*Direct SQL Passthru*	Full support for the long DECIMAL datatype. SET CONVERSION can alter the length and scale of numeric columns returned by a SELECT request.

Chapter/Appendix		**New Features**
11	*Environmental Commands*	SET BINDOPTIONS controls DB2 BIND parameters in static TABLE or MODIFY procedures. SET CURRENT DEGREE supports parallel processing in DB2 dynamic SQL requests. SET ISOLATION command supported for DB2. Available as of FOCUS 7.0.8 SET IXSPACE can override the default index space parameters for indexes created by the FOCUS CREATE FILE command or HOLD FORMAT DB2. SET OPTIFTHENELSE optimizes IF-THEN-ELSE DEFINE expressions (DB2 Interface only). Available as of FOCUS 7.0.8. SET PASSRECS controls the messages generated by successful Direct SQL Passthru UPDATE and DELETE commands. SET SQLJOIN OUTER controls outer join optimization. Available as of FOCUS 7.0.9. New Dialogue Manager variables store the release and version of DB2 or SQL/DS returned by the CONNECT command.
12	*Maintaining Tables With FOCUS*	Interface support for the FOCUS MAINTAIN facility. Interface segment activation logic in MODIFY write operations is now identical to that of standard FOCUS.
13	*Static SQL*	The Static SQL for TABLE facility, including: • The Interface SQL COMPILE and RUN facilities. • SQL host variable length considerations. • Verification and control of host variable placement.
F	*Additional DB2 Topics*	DRDA support, including: • SET CONNECTION. • RELEASE. • SET CURRENT PACKAGESET.

Related Publications

Related publications include:

- *FOCUS for IBM Mainframe User's Manual Release 7.0* (DN1000983.0495) and *FOCUS for IBM Mainframe MAINTAIN User's Manual Release 7.0* (DN1001000.0495). Available together as a documentation set (DN1001005.0495).
- *FOCUS for IBM Mainframe DB2 and SQL/DS Read/Write Interface Installation Guide Release 7.0* (DN1000937.0997).
- *FOCUS for IBM Mainframe Multi-Session Option Installation and Technical Reference Guide Release 7.0* (DN1000966.1095).
- *FOCUS for IBM Mainframe Summary of New Features Release 7.0.8R CD* (DN1001050.0698).
- Technical Memo TM7922, *DB2 Interface BIND Parameter Settings*.
- *IMS/DC Interface User's Manual* (DN1000050.0185).
- *FOCUS for IBM Mainframe IMS/DB Interface User's Manual and Installation Guide Release 7.0* (DN1000977.0997).
- *Cross-Machine Interface User's Manual* (DN0500005.0691), for information on the XMI/DB2 and XMI/SQL Interfaces.
- *FOCUS for IBM Mainframe User-Written Subroutines Library* (DN1000026.0398).

Note: The title and Document Number (DN) information provided here are accurate as of this printing. To ensure up-to-date information when ordering, please consult the latest *Information Builders® Publications Catalog*.

Information Builders Systems Journal

The *Information Builders Systems Journal* is a unique technical publication dedicated to providing you with the latest information necessary to enhance your use of Information Builders products.

Through its detailed articles, illustrated with code, screen shots, and other visuals, the Journal challenges you to develop better reporting habits, customize features to enhance your systems applications, utilize its tips and techniques for better performance and productivity, and so much more.

You can order the *Information Builders Systems Journal* from the Publications Catalog or from our World Wide Web site, http://www.ibi.com.

Terminology

The following summarizes standard terminology used throughout this manual:

- **SQL** refers to the Structured Query Language.
- **DB2** refers to the MVS® relational database management system (RDBMS).
- **SQL/DS** refers to the VM RDBMS.
- **RDBMS** is a generic term for either DB2 or SQL/DS. Where necessary, they are treated separately.
- **Table** is a generic term for a DB2 or SQL/DS table or view.
- **Interface** refers to both Interface products. Where necessary, the DB2 and SQL/DS Interfaces are treated separately.
- **Primary key** refers to the column or columns that uniquely identify a row in a table. Used together, the terms **primary key** and **foreign key** refer to the columns that relate two tables. In either case, they do not refer to primary and foreign keys as defined in SQL CREATE TABLE statements (RDBMS referential integrity) unless explicitly stated.
- **TABLE request** refers to a FOCUS report request.
- **Report request** refers to a FOCUS report request or Direct SQL Passthru report request.
- **FOCEXEC** refers to a FOCUS stored procedure or Direct SQL Passthru procedure.

Customer Support

Do you have questions about FOCUS, FOCUS products, or EDA?

Call Information Builders Customer Support Services (CSS) at (800) 736-6130 or (212) 736-6130. Customer Support Consultants are available Monday through Friday between 8:00 a.m. and 8:00 p.m. EST to address all your FOCUS or EDA questions. Information Builders consultants can also give you general guidance regarding product capabilities and documentation. Please be ready to provide your six-digit site code number (xxxx.xx) when you call.

You can also access support services electronically, 24 hours a day, with InfoResponse Online. InfoResponse Online is accessible through our World Wide Web site, http://www.ibi.com. It connects you to the tracking system and known-problem database at the Information Builders support center. Registered users can open, update, and view the status of cases in the tracking system, and read descriptions of reported software issues. New users can register immediately for this service. The technical support section of www.ibi.com also provides usage techniques, diagnostic tips, and answers to frequently asked questions.

To learn about the full range of available support services, ask your Information Builders representative about InfoResponse, or call (800) 969-INFO.To help our consultants answer your queries most effectively, please be ready to provide the following information when you call:

Your six digit site code number (xxxx-xx).

The FOCEXEC procedure (preferably with line numbers).

Master file with picture (provided by CHECK FILE).

Run sheet (beginning at login, including call to FOCUS), containing:

? RELEASE.

? FDT.

? LET.

? LOAD.

? COMBINE.

? JOIN.

? DEFINE.

? STAT.

? SET/? SET GRAPH.

? USE.

? TSO DDNAME OR CMS FILEDEF

The exact nature of the problem: for example, are the results or is the format incorrect; does an abend occur; are the text or calculations missing or misplaced; is this related to any other problem?

Has the procedure ever worked in its present form? Has it been changed recently?

What release of the operating system are you using? Has it, FOCUS, or an interface system changed?

Is this problem repeatable?

Information Builders consultants can also give you general guidance on FOCUS capabilities and documentation.

You can also FAX your questions to CSS at (212) 564-1881, or upload them to the FOCWIZARD or FOCSERVICES forums on Compuserve.

User Feedback

In an effort to produce effective documentation, the Documentation Services staff at Information Builders welcomes any opinion you can offer regarding this manual. Please use the Reader Comments form at the end of this manual to relay suggestions for improving the publication or to alert us to corrections. You can also use the Document Enhancement Request Form on our Web site, http://www.ibi.com.

Thank you, in advance, for your comments.

1 Introduction

Topics:

- FOCUS and RDBMS Interaction
- Environment
- Ease of Use
- Efficiency
- Security

With the FOCUS DB2 and SQL/DS Interface, you can use FOCUS to access DB2 and SQL/DS tables and views. FOCUS is well adapted to the DB2 and SQL/DS environments and fully supports the relational data model.

Beginners as well as advanced data processing professionals can take advantage of Interface data retrieval and analysis facilities, including easy-to-use, menu-driven query tools and a powerful reporting language that can satisfy virtually any requirement. FOCUS file descriptions integrate all facilities and provide transparent access to the underlying relational data structure.

The Interface also provides transaction processing and application development tools for updating and maintaining tables resident on the relational database management system (RDBMS). The FOCUS transaction processors, MAINTAIN and MODIFY, support interactive maintenance procedures as well as batch maintenance using external input files.

The Interface can manage multi-table, multi-record processing supported by all the constructs of a complete programming language (for example, PERFORM groups, computations, data value validations, error handling, GOTO statements, and IF-THEN-ELSE logic). It permits you to update multiple tables in a single procedure. It respects all RDBMS referential integrity rules while, optionally, maintaining its own referential integrity restrictions that you can specify within the update procedures themselves or in FOCUS Master Files.

Additionally, you can embed SQL statements (the language required by the RDBMS) directly into FOCUS application code to take advantage of SQL commands that control updates and locks (COMMIT WORK, ROLLBACK WORK), create RDBMS objects (CREATE INDEX, CREATE TABLE), and define security privileges (GRANT, REVOKE). With the Direct SQL Passthru facility, you can even include SQL SELECT statements in report requests and retrieve answer sets from the RDBMS. A Dialogue Manager variable indicates the success or failure of each SQL command. You can create new tables with SQL commands or with the FOCUS CREATE FILE command.

FOCUS complements RDBMS security features by permitting controlled data access at the user, table, field, or field-value levels. You can use FOCUS security to enhance existing RDBMS security.

The Interface translates FOCUS retrieval and update requests into an equivalent set of SQL statements. It also initiates and monitors communication between itself and the RDBMS, and provides descriptive error message and recovery support when necessary.

The two distinct components of the Interface are Read and Write:

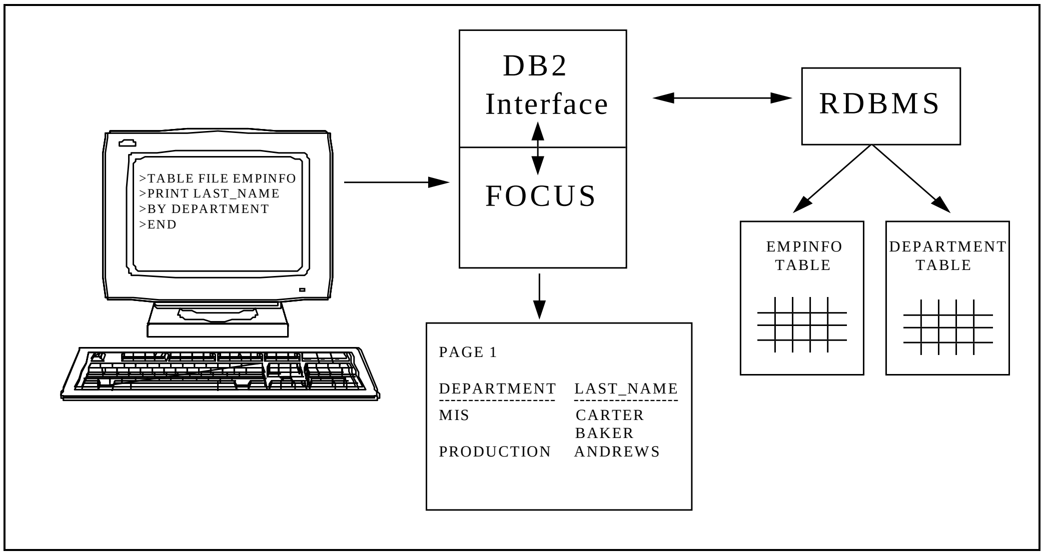
- The Read component translates FOCUS retrieval requests (such as TABLE and GRAPH) into SQL statements that define the request to the RDBMS. When the RDBMS returns data to the Interface in response to these statements, the Interface passes the data to the FOCUS Report Writer. The Report Writer can process data from any FOCUS-readable file.
- The Write component generates SQL statements from standard FOCUS MAINTAIN or MODIFY requests that retrieve and maintain data stored in RDBMS tables.

FOCUS and RDBMS Interaction

FOCUS and the RDBMS interact as follows:

1. Given a TABLE, MAINTAIN, or MODIFY request, the Interface builds SQL statements that define the request in terms the RDBMS can understand.
2. Having received these SQL statements from the Interface, the RDBMS retrieves or updates data targeted by the request.
3. In response, the RDBMS sends the data (answer set) and/or return code back to the Interface, which in turn passes it to FOCUS for further processing (for example, CRTFORM display and value changes).

The following diagram illustrates FOCUS interacting with a DB2 database:



The Interface functions as an RDBMS application that normally executes dynamic SQL. As such, it must be registered with the RDBMS in the same way as any other application.

The Interface as an RDBMS Application

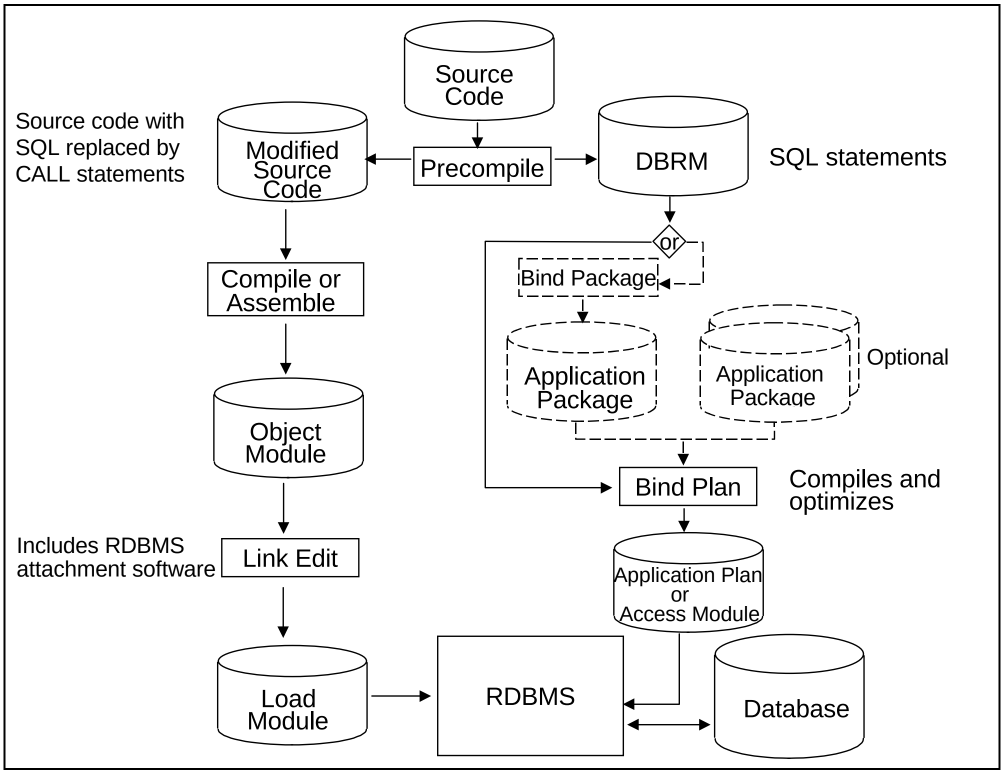
All applications that access RDBMS tables go through a precompile procedure prior to normal compilation. The precompiler copies all of the SQL statements from the application into a separate module called a Database Request Module (DBRM). It also modifies the original application program by transforming its SQL statements into comments and replacing them with calls to the DBRM. The modified application program can then go through the usual process for creating an executable load module.

The SQL statements from the original application program must also be made executable. The DBRM undergoes its own compilation and optimization process called a **bind**. The bind optimizes the SQL code, performs security checking, and determines the most efficient **access path** to the required data (the access path identifies available resources, such as specific indexes and scan methodologies for traversing the database). The result of the bind is called an Application Plan in DB2 and an Application Package in SQL/DS.

There are three ways of binding DBRMs into an Application Plan. One is to bind all of the DBRMs for an application into one large plan; the second is to bind each DBRM into a separate plan. The third is to bind separate DBRMs into small Application Packages which you then bind into a special type of Application Plan that consists of pointers to each Application Package. The advantage of this alternative is that you can re-bind individual packages without having to re-bind the entire Application Plan. In a distributed database environment, Application Packages are essential.

During execution, the program load module works in conjunction with the Application Plan or Access Module under control of the RDBMS.

The following diagram illustrates the process of preparing an application for execution:



You can use either dynamic or static SQL in TABLE and MODIFY requests. Dynamic SQL is the default for the Interface; however, even to use static SQL, you must have the Interface installed. The choice of dynamic or static SQL has the following effects on compilation and bind processing:

- With dynamic SQL, the Interface installation includes the precompile, compile, link-edit, and bind (Interface installation produces an Application Plan or Application Package for the Interface). The source program for the precompiler is an Assembler program that the Interface calls when it needs to execute SQL statements.
- With static SQL, you issue a FOCUS command to convert your FOCEXEC into a source program for the precompiler. The source program contains the SQL that would be generated by the FOCEXEC.

  At run time, the Interface executes static MODIFY procedures without further processing.

However, prior to executing a static TABLE request, the Interface re-generates the SQL in the request. It then compares the current SQL to the SQL generated during compilation. If the SQL has not changed, the Interface executes the FOCEXEC. Because of this extra step, you can run an altered TABLE request without re-binding it, as long as the SQL generated by the TABLE request has remained the same.

Chapter 13, *Static SQL*, discusses the advantages of static SQL and describes static module creation.

Environment

The Interface operates in conjunction with FOCUS to access:

- DB2 under MVS/ESA™, teleprocessing monitor environments (TSO, IMS/DC), the FOCUS Cross-Machine Interface (XMI), FOCUS Multi-Session Option (MSO), and batch.
- SQL/DS (also known as DB2/VM) under VM/CMS, the FOCUS Cross-Machine Interface (XMI) under DOS/VSE, and batch.

The Interface is compatible with all currently supported IBM DB2 and SQL/DS releases except where noted. In addition to DB2 and SQL/DS tables, FOCUS accesses FOCUS databases, QSAM and VSAM files, and, with the appropriate Interfaces installed, many other data sources such as IMS™, SUPRA®, TOTAL®, Oracle®, CA-IDMS®/DB, CA-Datacom®/DB, Teradata®, ADABAS®, System 2000, and Model 204®.

Ease of Use

With the DB2 or SQL/DS Interface installed, you can use the FOCUS language to request access to RDBMS tables and views. There is no need for specialized subroutines or embedded SQL commands, although with the Direct SQL Passthru facility you can include all SQL commands in report requests.

To make a table intelligible to FOCUS, describe each table or view once in FOCUS terminology. FOCUS stores this description as a Master File and an associated Access File. Once you create Master and Access Files for an RDBMS table, you can refer to the individual columns of the table via the FOCUS fieldname or the RDBMS column name. The AUTODB2 or AUTOSQL facility can create Master and Access Files for you automatically. (Issuing requests through the Direct SQL Passthru facility eliminates the need for FOCUS Master and Access Files but retains all FOCUS reporting capabilities.)

Once you have a FOCUS Master and Access File for a table, you can use all FOCUS facilities available at your site, such as the Report Writer, the database maintenance language, graphics, and statistics. You do not need to know SQL.

Efficiency

The Interface retrieves from the RDBMS only those rows or columns referenced in the FOCUS report request. Additionally, the Interface may pass to the RDBMS all of the work required to join, sort, and aggregate data. This reduces the volume of RDBMS-to-FOCUS communication, resulting in faster response times for Interface users.

The FOCUS DB2 Interface fully supports the DB2 Distributed Data Facility. It appends a “FOR FETCH ONLY” clause to SELECT statements passed to DB2. This clause assists the DB2 optimizer in access path selection, and offers significant performance improvement in the distributed database environment. In addition, the Interface supports the IBM Distributed Relational Database Architecture (DRDA). See Appendix F, *Additional DB2 Topics*, for a description of Interface DDF and DRDA support.

Security

FOCUS respects all existing RDBMS security. That is, an Interface user must be authorized, in DB2 or SQL/DS terms, to retrieve data or update tables. This authorization must come from the RDBMS database administrator or another authorized user.

FOCUS also provides its own security facilities; you can use them as a complement to RDBMS security. FOCUS security can enforce the following levels of restriction:

- File-level security to prevent access to a table.
- Field-level security to limit the fields within a table that are accessible to a user.
- Field-value security to limit the rows within a table that are accessible to a user based on the specified field's values.

Refer to the *FOCUS for IBM Mainframe User’s Manual* for information on FOCUS DBA security.

2 Getting Started

Topics:

- Getting Started Under MVS
- Getting Started Under VM/CMS
- Issuing Commands

This chapter explains how to invoke the Interface for a specific environment. It contains:

- Sample CLISTs, batch JCL, and required DDNAMEs for the DB2 Interface. (See *Getting Started Under MVS* on page 2-1.)
- A general list of steps, a sample initialization EXEC, and required filetype conventions for the SQL/DS Interface. (See *Getting Started Under VM/CMS* on page 2-8.)
- The Interface SQL ? command for displaying current session defaults. (See *Issuing Commands* on page 2-9.) Chapter 11, *Environmental Commands*, describes environmental commands for changing Interface defaults.

Note: The samples provided in this chapter do not contain site-specific information. Please check with your database administrator for MVS data set high-level qualifiers, passwords, proper authorization, or techniques for running FOCUS in VM batch.

Getting Started Under MVS

In the MVS operating environment, TSO or MSO controls the DB2 Interface and the FOCUS session. The Interface can access DB2 tables interactively from TSO or MSO, with TSO batch processing, or as an MVS batch job. You must know whether the Interface was installed with the Call Attachment Facility (CAF) or TSO Attachment; the CLIST requirements are different in each case.

You must also know the four-character subsystem identifier (SSID) of the DB2 subsystem you will access and the plan name assigned to the DB2 Interface when it was installed. The defaults for the SSID and plan values are DSN and DSQL respectively, unless your site changed these defaults at installation time.

If the Interface was installed with the Call Attachment Facility (CAF), you can use the SET SSID and SET PLAN environmental commands to specify the SSID and plan name from within FOCUS (see Chapter 11, *Environmental Commands*). For non-CAF installations, your CLIST specifies SSID and PLAN when it invokes the DSN command processor.

After you prepare your CLIST or JCL, ask your database administrator whether you require SELECT, INSERT, and/or UPDATE privileges for the tables or views you wish to access.

Note: The discussions in this section assume that all of your site's FOCUS and Interface libraries are catalogued under the same MVS high-level qualifier. The examples throughout this section use the identifier *prefix* to refer to this high-level qualifier.

Interactive Access From TSO

To run FOCUS interactively, you invoke a CLIST or REXX EXEC from TSO using standard FOCUS allocations for DDNAMEs FOCEXEC, ERRORS, and MASTER. Also include allocations for DDNAMEs FOCSQL, USERLIB, and FOCLIB. Allocate FOCSQL to the library containing FOCUS Access Files; allocate FOCLIB to the FOCUS product load library. Add the allocations for the FOCUS and Interface load libraries to USERLIB.

You can allocate the FOCUS and Interface load libraries (both from the same release and Maintenance Level of FOCUS) directly in your CLIST or REXX EXEC, or your site may choose to allocate them to DDNAME STEPLIB in your TSO logon procedure. (The *FOCUS for IBM Mainframe MVS/TSO Installation Guide* discusses the FOCUS CLIST in greater detail.)

Note: To display the current FOCUS release and Maintenance Level, issue the ? RELEASE query command at the FOCUS prompt.

If the FOCUS load libraries are not allocated in your TSO logon procedure, use the following CLIST (provided as member DB2EXCLI of the 'prefix.FOCSQL.DATA' data set) after editing it to conform to your site's standards. This CLIST assumes that the Interface was installed with the Call Attachment Facility (CAF)

```
PROC 0
CONTROL MSG NOLIST NOFLUSH
ALLOC F(FOCEXEC) DA('user.FOCEXEC.DATA' -
                    'prefix.FOCEXEC.DATA')  SHR REUSE
ALLOC F(MASTER)  DA('user.MASTER.DATA' -
                    'prefix.MASTER.DATA')   SHR REUSE
ALLOC F(FOCSQL)  DA('user.FOCSQL.DATA' -
                    'prefix.FOCSQL.DATA')   SHR REUSE
ALLOC F(TRF)     DA('user.TRF.DATA' -
                    'prefix.FOCSQL.DATA')   SHR REUSE
ALLOC F(ERRORS)  DA('prefix.ERRORS.DATA')   SHR REUSE
ALLOC F(USERLIB) DA('prefix.FOCSQL.LOAD' -
                    'prefix.FOCLIB.LOAD' -
                    'prefix.FUSELIB.LOAD')  SHR REUSE
ALLOC F(FOCLIB)  DA('prefix.FOCLIB.LOAD')   SHR REUSE
CALL  'prefix.FOCLIB.LOAD(FOCUS)'
```

where:

prefix

Is the high-level qualifier for your site's FOCUS production libraries.

user

Is the high-level qualifier for your private version of a library.

Note: The allocation for DDNAME TRF (a FOCUS Window Transfer File PDS) is included because it is required for the FOCUS EXPLAIN utility. The EXPLAIN utility is discussed in Chapter 7, *The Interface Optimizer*.

If the Interface was installed to use TSO Attachment, replace the CALL statement in the previous CLIST with the following lines

```
DSN SYSTEM(ssid)
RUN PROGRAM(FOCUS)  PLAN(plan) LIBRARY('prefix.FOCLIB.LOAD')
END
```

where:

ssid

Is the DB2 subsystem identifier.

plan

Is the name of the Interface Application Plan.

prefix

Is the high-level qualifier for your site's FOCUS production libraries.

Your DB2 database administrator can supply the parameters and the attachment facility chosen at the time the Interface was installed.

If you want to allocate the FOCUS and Interface load libraries in the TSO logon procedure, use the following JCL as a model. Alter the preceding CLIST to eliminate the allocations for DDNAMEs FOCLIB and USERLIB.

Concatenate the libraries to DDNAME STEPLIB as in the following sample:

```
//LOGON     EXEC  PGM=IKJEFT01,DYNAMNBR=25
//STEPLIB   DD    DSN=prefix.FOCSQL.LOAD,DISP=SHR
//          DD    DSN=prefix.FOCLIB.LOAD,DISP=SHR
//          DD    DSN=prefix.FUSELIB.LOAD,DISP=SHR
//          DD    DSN=DSN510.SDSNLOAD,DISP=SHR
```

Note: The 'prefix.FOCSQL.DATA' data set includes sample batch JCL in members DB2EXJCA (for CAF) and DB2EXJCT (for TSO Attachment).

Interactive Access From MSO

To access the Interface from MSO, you must allocate the Interface files using DYNAM commands. You can include these allocations in a FOCEXEC.

The following FOCEXEC example allocates the Interface files

```
-* DDNAME FOCEXEC
DYNAM ALLOC FILE FOCEXEC DA user.FOCEXEC.DATA SHR REUSE
DYNAM ALLOC FILE FOCPROG DA prefix.FOCEXEC.DATA SHR REUSE
DYNAM CONCAT FILE FOCEXEC FOCPROG
-* DDNAME MASTER
DYNAM ALLOC FILE MASTER DA user.MASTER.DATA SHR REUSE
DYNAM ALLOC FILE FOCMAST DA prefix.MASTER.DATA SHR REUSE
DYNAM CONCAT FILE MASTER FOCMAST
-* DDNAME FOCSQL
DYNAM ALLOC FILE FOCSQL DA user.FOCSQL.DATA SHR REUSE
DYNAM ALLOC FILE FOCAFD DA prefix.FOCSQL.DATA SHR REUSE
DYNAM CONCAT FILE FOCSQL FOCAFD
-* DDNAME TRF
DYNAM ALLOC FILE TRF DA user.TRF.DATA SHR REUSE
DYNAM ALLOC FILE FOCTRF DA prefix.FOCSQL.DATA SHR REUSE
DYNAM CONCAT FILE TRF FOCTRF
```

where:

user
Is the high-level qualifier for your private version of a library.

prefix
Is the high-level qualifier for your site’s FOCUS production libraries.

Note:

- The MSO and Interface load libraries must be from the same release and Maintenance level of FOCUS.
- The allocation for DDNAME TRF (a FOCUS Window Transfer File PDS) is included because it is required for the FOCUS EXPLAIN utility (see Chapter 7, *The Interface Optimizer*).

Batch Access

For batch access, you must submit JCL that allocates the libraries required by FOCUS. If the Interface was installed with the Call Attachment Facility, you can use the following sample JCL (provided as member DB2EXJCA of the ‘prefix.FOCSQL.DATA’ data set) after editing it to conform to your site’s standards and adding a JOB card

```
//JOB card goes here
//BATCAF   EXEC PGM=FOCUS
//STEPLIB  DD   DISP=SHR,DSN=prefix.FOCSQL.LOAD
//         DD   DISP=SHR,DSN=prefix.FOCLIB.LOAD
//         DD   DISP=SHR,DSN=prefix.FUSELIB.LOAD
//         DD   DISP=SHR,DSN=DSN510.SDSNLOAD
//ERRORS   DD   DISP=SHR,DSN=prefix.ERRORS.DATA
//FOCEXEC  DD   DISP=SHR,DSN=user.FOCEXEC.DATA
//         DD   DISP=SHR,DSN=prefix.FOCEXEC.DATA
//MASTER   DD   DISP=SHR,DSN=user.MASTER.DATA
//         DD   DISP=SHR,DSN=prefix.MASTER.DATA
//FOCSQL   DD   DISP=SHR,DSN=user.FOCSQL.DATA
//         DD   DISP=SHR,DSN=prefix.FOCSQL.DATA
//FOCSTACK DD   UNIT=SYSDA,SPACE=(TRK,(5,5))
//FOCSORT  DD   UNIT=SYSDA,SPACE=(TRK,(5,5))
//HOLD     DD   UNIT=SYSDA,SPACE=(TRK,(5,5))
//HOLDMAST DD   UNIT=SYSDA,SPACE=(TRK,(5,5,5))
//HOLDACC  DD   UNIT=SYSDA,SPACE=(TRK,(5,5,5))
//OFFLINE  DD   SYSOUT=A
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   *
SQL DB2 SET SSID ssid
SQL DB2 SET PLAN plan
TABLE FILE db2table
        .
        .
        .
END
FIN
/*
```

where:

prefix

Is the high-level qualifier for your site’s FOCUS production libraries.

user

Is the high-level qualifier for your private version of a library.

ssid

Is the DB2 subsystem identifier, used to access a subsystem other than the default.

plan

Is the name of the Interface Application Plan.

db2table

Is the name of the Master File for the DB2 table or view.

Note: The need for Interface SET commands depends on the particular application.

If the Interface was installed to use the TSO Attachment Facility, execute the TSO batch program, IKJEFT01, which in turn calls FOCUS. You can use the following sample JCL (provided as member DB2EXJCT of the 'prefix.FOCSQL.DATA' data set) after editing it to conform to your site's standards and adding a JOB card

```
//JOB card goes here
//BATTSO   EXEC PGM=IKJEFT01
//STEPLIB  DD   DISP=SHR,DSN=prefix.FOCSQL.LOAD
//         DD   DISP=SHR,DSN=prefix.FOCLIB.LOAD
//         DD   DISP=SHR,DSN=prefix.FUSELIB.LOAD
//         DD   DISP=SHR,DSN=DSN510.SDSNLOAD
//ERRORS   DD   DISP=SHR,DSN=prefix.ERRORS.DATA
//FOCEXEC  DD   DISP=SHR,DSN=user.FOCEXEC.DATA
//         DD   DISP=SHR,DSN=prefix.FOCEXEC.DATA
//MASTER   DD   DISP=SHR,DSN=user.MASTER.DATA
//         DD   DISP=SHR,DSN=prefix.MASTER.DATA
//FOCSQL   DD   DISP=SHR,DSN=user.FOCSQL.DATA
//         DD   DISP=SHR,DSN=prefix.FOCSQL.DATA
//FOCSTACK DD   UNIT=SYSDA,SPACE=(TRK,(5,5))
//FOCSORT  DD   UNIT=SYSDA,SPACE=(TRK,(5,5))
//HOLD     DD   UNIT=SYSDA,SPACE=(TRK,(5,5))
//HOLDMAST DD   UNIT=SYSDA,SPACE=(TRK,(5,5,5))
//HOLDACC  DD   UNIT=SYSDA,SPACE=(TRK,(5,5,5))
//OFFLINE  DD   SYSOUT=A
//SYSPRINT DD   SYSOUT=A
//SYSTSPRT DD   SYSOUT=A
//SYSTSIN  DD   *
   DSN SYSTEM(ssid)
   RUN PROGRAM(FOCUS) PLAN(plan) LIBRARY('prefix.FOCLIB.LOAD')
   END
//SYSIN    DD   *
  TABLE FILE db2table
    .
    .
    .
END
FIN
/*
```

where:

prefix

Is the high-level qualifier for your site's FOCUS production libraries.

user

Is the high-level qualifier for the your private version of a library.

ssid

Is the DB2 subsystem identifier, used to access a subsystem other than the default.

plan

Is the name of the Interface Application Plan.

db2table

Is the name of the Master File for the DB2 table or view.

The file allocations in batch are similar to those in the CLIST, with a few changes. The STEPLIB allocation replaces the FOCLIB and USERLIB allocations from the CLIST. You must allocate the file containing executable FOCUS commands to DDNAME SYSIN. The FOCUS commands are coded “in-stream” in the sample; however they could have been stored in a data set. Output is written to the file or SYSOUT class allocated to DDNAME SYSPRINT. The FIN command is required to terminate FOCUS.

For proper JOB card specifications and data set names for your site, consult with your system support staff. For additional information on FOCUS and batch processing, refer to the *FOCUS for IBM Mainframe User’s Manual*.

Additional Prerequisites: File Descriptions

The Interface requires a Master and Access File for each RDBMS table referenced by FOCUS. In MVS, file descriptions and FOCEXECs are stored as members of partitioned data sets (PDSs). The partitioned data sets are allocated to the following DDNAMEs:

DDNAME	**Contents (PDS Members)**
MASTER	Master Files.
FOCSQL	Access Files.
FOCEXEC	FOCUS procedures.

Execute the AUTODB2 FOCEXEC, supplied with the Interface, to automatically create Master and Access Files for existing RDBMS tables. You can customize the resulting descriptions with a text editor. Chapter 6, *Automated Procedures*, describes step-by-step instructions for AUTODB2.

You can create new table definitions in the RDBMS from the FOCUS environment by issuing either the FOCUS CREATE FILE command (after creating a Master File and an Access File) or the SQL CREATE TABLE command. See Chapter 6, *Automated Procedures*, for the FOCUS CREATE FILE command.

Getting Started Under VM/CMS

In the VM operating environment, the CMS teleprocessing monitor controls the Interface and FOCUS session. The Interface can access SQL/DS tables interactively. Techniques for running FOCUS in batch CMS may also be available at your site; those techniques are not discussed here. In addition to SQL/DS tables, you can access VSAM files, FOCUS databases, and other file types with the appropriate FOCUS Interfaces installed.

Interactive Access From CMS

The following is a general outline of steps for invoking the SQL/DS Interface. The details vary from site to site:

1. After logging on to VM, link to and access the FOCUS production disk.

2. Execute an initialization EXEC.

The EXEC should link to the SQL/DS production disk and execute SQLINIT, an IBM EXEC. The SQLINIT procedure creates bootstrap modules for the appropriate SQL/DS database and places them on the disk accessed as filemode A. (**Note:** You do not have to rerun SQLINIT unless you want access to a different SQL/DS database. You may wish to have your EXEC check for the existence of the bootstrap modules and conditionally execute SQLINIT if they do not exist.)

The following is a sample procedure

```
CP LINK SQLDBA 195 485 RR password
ACCESS 485 Q
EX SQLINIT DB(dbname)
```

where:

password
Is the read password for the SQL/DS production disk.

dbname
Is the name of the SQL/DS database.

You can issue these commands interactively before entering the FOCUS environment, but it is probably more convenient to include them in an EXEC that issues them automatically.

3. Invoke FOCUS. If your userid is authorized to CONNECT to SQL/DS, no further action is required. In some cases, you may need to issue the SQL/DS CONNECT command. Refer to Chapter 3, *Security*, for information on CONNECT syntax and usage.

Additional Prerequisites: File Descriptions

The Interface requires a Master and Access File for each SQL/DS table referenced by FOCUS. In VM/CMS, file descriptions and FOCEXECs exist as separate files. The naming conventions are:

Filetype	**Contents**
MASTER	Master Files.
FOCSQL	Access Files.
FOCEXEC	FOCUS procedures.

Execute the AUTOSQL EXEC, supplied with the Interface, to automatically create Master and Access Files for existing SQL/DS tables. You can customize the resulting descriptions with a text editor. Chapter 6, *Automated Procedures*, describes step-by-step instructions for AUTOSQL.

You can create new table definitions from the FOCUS environment by issuing either the FOCUS CREATE FILE command (after creating a Master File and an Access File) or the SQL CREATE TABLE command. See Chapter 6, *Automated Procedures*, for the FOCUS CREATE FILE command.

Issuing Commands

Direct SQL Passthru is a facility for passing native SQL commands directly to the RDBMS without intervention by FOCUS, and for issuing environmental commands that display or change Interface default settings. A brief introduction follows. For a detailed discussion of the Direct SQL Passthru facility, consult Chapter 9, *Direct SQL Passthru*. Native SQL commands are discussed in Appendix B, *Native SQL Commands*.

Direct SQL Passthru

You can invoke Interface environmental commands or pass any native SQL command directly to the RDBMS with Direct SQL Passthru.

Inform the Interface which RDBMS to access using one of the following techniques:

- Include a qualifier (DB2 or SQLDS) after the SQL keyword in each command. For example

  ```
  SQL DB2 INSERT INTO DPBRANCH VALUES (1,'WEST','PIAF','NY') ;
  END
  ```

- Set the default database engine once with the SET SQLENGINE command. The Interface automatically passes subsequent SQL statements to that RDBMS. For example

  ```
  SET SQLENGINE=DB2
  SQL INSERT INTO DPBRANCH VALUES (1,'WEST','PIAF','NY') ;
  END
  ```

  You can reset the default engine at any time during the FOCUS session.

Native SQL

Direct SQL Passthru is the preferred method for issuing your own SQL commands through the Interface. If necessary, however, you can issue Interface environmental commands and pass many SQL commands directly to the RDBMS without the Passthru facility.

To invoke native SQL without Direct SQL Passthru, prefix each command with an environmental qualifier (MVS, TSO, or CMS). For example:

```
MVS SQL SET AUTOCLOSE ON
```

The qualifiers MVS and TSO are interchangeable. You may not issue SQL commands that return data (such as SELECT commands) with this method.

For the remainder of this manual, assume Direct SQL Passthru is in effect unless otherwise indicated.

Interface Environmental Commands

The Interface provides environmental commands that display or change Interface default settings for the duration of the FOCUS session. You can issue these commands from the FOCUS command line or include them in a FOCEXEC.

To display the current Interface settings for a DB2 session within FOCUS, issue the following Interface query command:

```
> sql db2 ?
(FOC1440) CURRENT SQL INTERFACE SETTINGS ARE :
(FOC1442) CALL ATTACH FACILITY IS             -  : ON
(FOC1447) SSID FOR CALL ATTACH IS             -  : DSN
(FOC1448) ACTIVE PLAN FOR CALL ATTACH IS      -  :
(FOC1459) USER SET PLAN FOR CALL ATTACH IS    -  :
(FOC1460) INSTALLATION DEFAULT PLAN     IS    -  : P7009701
(FOC1503) SQL STATIC OPTION IS                -  : OFF
(FOC1444) AUTOCLOSE OPTION IS                 -  : ON FIN
(FOC1496) AUTODISCONNECT OPTION IS            -  : ON FIN
(FOC1499) AUTOCOMMIT OPTION IS                -  : ON COMMAND
(FOC1446) DEFAULT DBSPACE IS                  -  :
(FOC1449) CURRENT SQLID IS                    -  : SYSTEM DEFAULT
(FOC1424) ISOLATION LEVEL FOR DB2 TABLE INTERFACE IS  :
(FOC1441) WRITE FUNCTIONALITY IS              -  : ON
(FOC1445) OPTIMIZATION OPTION IS              -  : ON
(FOC1484) SQL ERROR MESSAGE TYPE IS           -  : FOCUS
(FOC1497) SQL EXPLAIN OPTION IS               -  : OFF
(FOC1552) INTERFACE DEFAULT DATE TYPE         -  : NEW
>  >
```

Chapter 11, *Environmental Commands*, explains Interface environmental commands for DB2 and SQL/DS.

3 Security

Topics:

- DB2 CURRENT SQLID
- SQL/DS CONNECT Authority
- SQL GRANT and REVOKE
- FOCUS DBA Security

In any computer system, it is important to secure data from unauthorized access. Both the RDBMS and FOCUS provide security mechanisms to ensure that users access only those objects for which they have authorization.

This chapter examines the DB2 SET CURRENT SQLID command, the SQL/DS CONNECT command, GRANT authorization for DB2 and SQL/DS users, and FOCUS security.

DB2 CURRENT SQLID

The DB2 RDBMS accepts two types of ID, the primary authorization ID and one or more optional secondary authorization IDs; it also recognizes the CURRENT SQLID setting.

Any interactive user or batch program that accesses a DB2 subsystem is identified by a primary authorization ID. A security system such as RACF® normally provides the ID to DB2. During the process of connecting to DB2, the primary authorization ID may be associated with one or more secondary authorization IDs (usually RACF groups). Each site controls whether it uses secondary authorization IDs. For more information about using the Interface in conjunction with external security packages, see the *FOCUS for IBM Mainframe Multi-Session Option Installation and Technical Reference Guide*.

The DB2 database administrator (DBA) may grant privileges to a secondary authorization ID that are not granted to the primary ID. Thus, secondary authorization IDs provide the means for granting the same privileges to a group of users. (The DBA associates individual primary IDs with a secondary ID and grants the privileges to the secondary ID.)

The DB2 CURRENT SQLID may be the primary authorization ID or any associated secondary authorization ID. At the beginning of the FOCUS session, the CURRENT SQLID is the primary authorization ID.

You can reset the CURRENT SQLID using the following Interface command

```
SQL [DB2] SET CURRENT SQLID = 'sqlid'
```

where:

`DB2`
: Is required if you did not previously issue the SET SQLENGINE command for DB2 (see Chapter 9, *Direct SQL Passthru*).

`sqlid`
: Is the desired primary or secondary authorization ID, enclosed in single quotation marks. All DB2 security rules are respected.

Unless you issue the SET OWNERID command described in *OWNERID* in Chapter 11, *Environmental Commands*, the CURRENT SQLID is the implicit owner for unqualified tablenames and the default owner ID for DB2 objects, such as tables or indices, created with dynamic SQL statements. (For example, the FOCUS CREATE FILE command issues dynamic SQL statements.) The CURRENT SQLID is also the sole authorization ID for GRANT and REVOKE statements. It must have all the privileges needed to create objects as well as GRANT and REVOKE privileges.

Other types of requests, such as FOCUS TABLE (SQL SELECT) and MODIFY (SQL SELECT, INSERT, UPDATE, or DELETE) requests, automatically search for the necessary authorization using the combined privileges of the primary authorization ID and all of its associated secondary authorization IDs, regardless of the DB2 CURRENT SQLID setting.

The CURRENT SQLID setting remains in effect until the communication thread to DB2 is disconnected, when it reverts to the primary authorization ID.

SQL/DS CONNECT Authority

CONNECT authority controls access to the SQL/DS RDBMS. FOCUS supports both implicit and explicit connection to SQL/DS.

With the implicit method, your VM userid automatically connects to SQL/DS when you access the Interface. This method requires that the SQL/DS database administrator (DBA) grant CONNECT privileges to either the VM userid or globally (to ALLUSERS). An implicit CONNECT is transparent; it requires no CONNECT command or password.

To CONNECT to SQL/DS explicitly, issue the following command after entering FOCUS

```
SQL [SQLDS] CONNECT [ sqlid IDENTIFIED BY password ] [TO dbname]
```

where:

`SQLDS`
: Is required if you did not previously issue the SET SQLENGINE command for SQL/DS (see Chapter 9, *Direct SQL Passthru*).

`sqlid`
: Is the optional SQL/DS individual or group authorization ID.

`password`
: Is the optional password associated with the SQL/DS authorization ID.

`dbname`
: Is the name of the SQL/DS database to which you want to connect. If you do not specify dbname, you connect to the default database established by the SQLINIT EXEC (see Chapter 2, *Getting Started*).

The CONNECT command allows you to assume the privileges of another SQL/DS authorization ID. The SQL/DS database administrator typically uses this technique to distribute privileges to a group of users. If the group connects as the same SQL/DS authorization ID, the DBA only needs to grant privileges to the group authorization ID and not to the members of the group.

You can include the CONNECT command in a FOCEXEC, such as the PROFILE FOCEXEC, and you can encrypt the FOCEXEC to prevent unauthorized viewing of the SQL/DS authorization ID and password.

SQL GRANT and REVOKE

The SQL GRANT and REVOKE commands control access to tables and views within the DB2 and SQL/DS environments. Without proper authorization, users are denied access to RDBMS tables. Only the RDBMS database administrator and the creator of the table or view implicitly hold these privileges.

The SQL GRANT command distributes privileges to others. The SELECT privilege authorizes the user to generate reports from a table. The INSERT, UPDATE, and DELETE privileges each provide specific capabilities for changing the values within a table. Contact your RDBMS database administrator for more details on RDBMS security.

FOCUS honors the security established by the SQL GRANT mechanism. For example, without SELECT access for the specified table, FOCUS cannot generate a requested report.

FOCUS DBA Security

You can implement FOCUS security as a complement to existing SQL GRANT security. FOCUS DBA security provides data control at a number of different levels. It can give users limited access to a specific table by restricting their access to particular columns and/or rows; it can further restrict their access to rows whose columns contain certain values. With this mechanism, a site can securely define a few global tables and eliminate the need to create a complex series of views for every combination of access rights.

To implement FOCUS security for a table, specify the security rules in the Master File following the description of the table. The FOCUS DBA (as defined in the Master File) can prevent unauthorized viewing of the restriction rules by encrypting the Master. For information regarding FOCUS DBA security or encryption, consult the *FOCUS for IBM Mainframe User's Manual.*

4 Describing Tables to FOCUS

Topics:

- Master Files
- Access Files
- The FOCUS OCCURS Segment

In order to access a table or view using FOCUS, you must first describe it to FOCUS in two files: a Master File, and an associated Access File.

Note: The term *table* in this manual refers to both RDBMS base tables and views.

The Master File describes the columns of the RDBMS table using keywords in comma-delimited format (see *Master Files* on page 4-2). The Access File includes additional parameters that complete the FOCUS definition of the RDBMS table (see *Access Files* on page 4-13). The Interface requires both descriptions to generate SQL queries.

There are two methods for creating Master and Access Files:

- Execute an automated procedure—AUTODB2 for MVS or AUTOSQL for VM—that creates Master and Access Files for existing RDBMS tables. Chapter 6, *Automated Procedures*, describes these facilities.
- Use an editor to manually create the descriptions of the table. Refer to a copy of the native SQL CREATE TABLE statement or a detailed report from the system catalog tables for column names, datatypes, lengths, and other descriptive information.

FOCUS file descriptions can represent an entire table or part of a table. Also, several pairs of file descriptions can define different subsets of columns for the same table, or one pair of Master and Access Files can describe several tables. This chapter presents single-table Master and Access Files. For multi-table file descriptions, see Chapter 5, *Multi-Table Structures*.

You can represent tables with repeating columns as FOCUS OCCURS segments. See *The FOCUS OCCURS Segment* on page 4-19 for an explanation of these virtual constructs.

FOCUS processes a request with the following steps:

1. It locates the Master File for the table. The filename specified in the report request identifies the Master File by its MVS membername or CMS filename.
2. It detects the SUFFIX value, SQLDS, in the Master File. Since this value indicates that the data is in a DB2 or SQL/DS table, FOCUS passes control to the Interface.

3. The Interface locates the corresponding Access File, uses the information contained in both descriptions to generate the SQL statements required by the report request, and passes the SQL statements to the RDBMS.

4. The Interface retrieves the answer sets generated by the RDBMS and returns control to FOCUS. Depending on the requirements of the request, FOCUS may perform additional processing on the returned data.

This example is a FOCUS Master File for the table EMPINFO:

```
FILENAME=EMPINFO       ,SUFFIX=SQLDS,$

SEGNAME=EMPINFO        ,SEGTYPE=S0,$
 FIELD=EMP_ID          ,ALIAS=EID        ,USAGE=A9    ,ACTUAL=A9,$
 FIELD=LAST_NAME       ,ALIAS=LN         ,USAGE=A15   ,ACTUAL=A15,$
 FIELD=FIRST_NAME      ,ALIAS=FN         ,USAGE=A10   ,ACTUAL=A10,$
 FIELD=HIRE_DATE       ,ALIAS=HDT        ,USAGE=YMD   ,ACTUAL=DATE,$
 FIELD=DEPARTMENT      ,ALIAS=DPT        ,USAGE=A10   ,ACTUAL=A10, MISSING=ON,$
 FIELD=CURRENT_SALARY  ,ALIAS=CSAL       ,USAGE=P9.2  ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE    ,ALIAS=CJC        ,USAGE=A3    ,ACTUAL=A3,$
 FIELD=ED_HRS          ,ALIAS=OJT        ,USAGE=F6.2  ,ACTUAL=F4, MISSING=ON,$
 FIELD=BONUS_PLAN      ,ALIAS=BONUS_PLAN ,USAGE=I4    ,ACTUAL=I4,$
```

The following is an Access File for the table EMPINFO:

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1,
 WRITE = YES,  DBSPACE = PUBLIC.SPACE0,$
```

Appendix C, *File Descriptions and Tables*, contains a complete list of Master Files and Access Files for the examples cited in this manual.

Master Files

A table is an RDBMS object consisting of rows and columns. A FOCUS Master File represents a table as a single segment. The syntax for describing an RDBMS table in a FOCUS Master File is similar to that for any external (non-FOCUS DBMS) file.

A Master File contains three types of declarations:

- The file declaration indicates that the data is stored in a DB2 or SQL/DS table or view.
- The segment declaration identifies a table.
- Field declarations describe the columns of the table.

Each declaration must begin on a separate line. A declaration consists of keyword-value pairs (called attributes) separated by commas. A declaration can span as many lines as necessary, as long as no single keyword-value pair spans two lines. Certain keywords are required; the rest are optional (see *Optional Field Attributes* on page 4-13).

Do not use system or SQL reserved words as names for files, segments, fields, or aliases. Specifying a reserved word generates the SQL syntax errors -104, -105, or -106.

File Attributes

Each Master File begins with a file declaration that names the file and describes the type of data source, in this case an RDBMS table or view. The file declaration has two attributes, FILENAME and SUFFIX. The syntax is

```
FILE[NAME]=name, SUFFIX=SQLDS [,$]
```

where:

name
: Is a one- to eight-character filename.

SQLDS
: Is the SUFFIX value for the DB2 and SQL/DS Interfaces.

FILENAME

The FILENAME (or FILE) keyword names the Master File. For DB2, the name of the Master File is its membername within the PDS allocated to DDNAME MASTER. For SQL/DS, the Master File name is the filename whose filetype is MASTER.

The Master File name consists of up to eight alphanumeric characters and must contain at least one letter. You should make the name representative of the table or view contents. It can have the same name as the RDBMS table if the table name complies with FOCUS naming conventions. It must have the same name as its corresponding Access File.

SUFFIX

The SUFFIX keyword indicates that the DB2 or SQL/DS Interface is required for interpreting requests. With either Interface, DB2 or SQL/DS, the SUFFIX value is SQLDS.

Segment Attributes

Each table described in a FOCUS Master File requires a segment declaration that consists of at least two attributes, SEGNAME and SEGTYPE.

If several Master Files (used only with TABLE requests) include the same table, you can avoid repeating the same description multiple times. Describe the table in *one* of the Master Files, and use the CRFILE attribute in the other Master Files to access the existing description. For a full explanation of remote segment descriptions, see *Remote Segment Descriptions* in Appendix A, *Additional Topics*.

The syntax for a segment declaration is

```
SEGNAME=segname, SEGTYPE={S0|KL} [,CRFILE=crfile] [,$]
```

where:

`segname`
: Is a one- to eight-character name that identifies the segment. If this segment references a remote segment description, segname must be identical to the SEGNAME from the Master File that contains the full definition of the RDBMS table's columns (see *Remote Segment Descriptions* in Appendix A, *Additional Topics*).

`S0`
: S zero indicates to the Interface that the RDBMS handles the storage order of the data.

`KL`
: References a remote segment description (see *Remote Segment Descriptions* in Appendix A, *Additional Topics*).

`crfile`
: Is required only to reference a remote segment description. Indicates the name of the remote Master File that contains the full definition of the RDBMS table's columns (see *Remote Segment Descriptions* in Appendix A, *Additional Topics*).

SEGNAME

The SEGNAME attribute identifies one table or view. The one- to eight-character SEGNAME value may be the same as the name chosen for FILENAME, the actual table name, or an arbitrary name. To reference a remote segment description, the SEGNAME value must be identical to the SEGNAME in the Master File that contains the full definition of the RDBMS table's columns.

The corresponding Access File must contain a segment declaration with the same SEGNAME value as the Master File. The segment declaration in the Access File specifies the name of the RDBMS table. In this manner, the SEGNAME value serves as a link to the actual table name.

SEGTYPE

In a single table Master File, SEGTYPE always has a value of S0 (or KL for a remote segment description). The RDBMS assumes responsibility for both physical storage of rows and the uniqueness of column values (if a unique index exists). SEGTYPE values for multi-table Master Files are discussed in *SEGTYPE* in Chapter 5, *Multi-Table Structures*.

CRFILE

Include the CRFILE attribute in a segment declaration only if the actual description of the table's columns is stored in another (remote) Master File. The CRFILE value must be the name of the Master File that contains the full definition of the RDBMS table's columns. For a complete discussion of remote segment descriptions, see *Remote Segment Descriptions* in Appendix A, *Additional Topics*.

Field Attributes

Each row in a table consists of one or more columns. In the Master File, you define each column as a FOCUS field with the primary attributes FIELDNAME, ALIAS, USAGE, ACTUAL, and MISSING. The *FOCUS for IBM Mainframe User's Manual* explains additional attributes.

You can get values for these attributes from the RDBMS system catalog table SYSCOLUMNS. Refer to *Querying the RDBMS SYSCOLUMNS Catalog* in Appendix A, *Additional Topics,* for a sample request.

The syntax for a field declaration is

```
FIELD[NAME]=name, [ALIAS=]sqlcolumn,  [{USAGE|FORMAT}=] display [,ACTUAL=]sqlfmt
   [, MISSING= {OFF|ON}]   ,$
```

where:

name
: Is a 1- to 66-character unqualified name. In requests, you can qualify a fieldname with its Master File and/or segment name. Although the qualifiers and qualification characters do not appear in the Master File, they count toward the 66-character maximum. *Long Fieldname Considerations* in Appendix A, *Additional Topics*, contains a discussion of long fieldnames. For more information, consult the *FOCUS for IBM Mainframe User's Manual*.

sqlcolumn
: Is the RDBMS column name, up to 18 characters long.

display
: Is the FOCUS display format for the field.

sqlfmt
: Is the FOCUS definition of the RDBMS datatype and length, in bytes, for the field. (See *ACTUAL Format Conversion Chart* on page 4-7.)

OFF|ON
: Indicates whether the field can contain null values. OFF, the default, does not permit null values.

The Primary Key

A table's primary key is the column or combination of columns whose values uniquely identify each row of the table. In the EMPINFO table, every employee is assigned a unique employee identification number. This identification number, and its corresponding employee, are represented by one (and only one) row of the table.

Note: The terms *primary key* and *foreign key* refer to columns that relate two tables. In this manual, they do not refer to primary and foreign keys defined in SQL CREATE TABLE statements (RDBMS referential integrity) unless explicitly stated.

The Interface uses information from both the Master File and the Access File to identify the primary key. In the Access File, the KEYS=n attribute specifies the *number* of key fields, n. In the Master File, the *first* n fields described immediately after the segment declaration constitute the primary key. Therefore, the order of field declarations in the Master File is significant.

To define the primary key in a Master File, describe its component fields first after the segment declaration. You can specify the remaining fields (those that do not participate in the primary key) in any order.

The KEYS attribute in the Access File completes the process of defining the primary key.

Typically, the primary key is supported by an SQL unique index to prevent the insertion of duplicate key values. The Interface itself does not require any index on columns comprising the primary key (although a unique index is certainly desirable for both data integrity and performance reasons).

Note: If the table was created with RDBMS referential integrity constraints (a PRIMARY KEY was specified in the SQL CREATE TABLE statement), then the RDBMS requires a unique index on that key. SQL/DS creates this index automatically. DB2 only flags the need for the index; it must be created prior to inserting any data. In either case, the RDBMS requires the index, not the Interface.

FIELDNAME

FOCUS fieldnames must be unique within a single-table Master File and can consist of up to 66 alphanumeric characters (including any filename and segname qualifiers and qualification characters you may later prefix to them in requests). Within the Master File, fieldnames cannot include qualifiers. Column names are acceptable values if they meet the following FOCUS naming conventions:

- A name can consist of letters, digits, and underscore characters. Special characters and embedded blanks are not advised.
- The name must contain at least one letter.

Since the fieldname appears as the default column title for reports, select a name that is representative of the data. In TABLE, GRAPH, and MODIFY requests, you can specify fieldnames, aliases, or a unique truncation of either. MAINTAIN does not support alias names or truncated names. In all requests, you can qualify a fieldname with its filename and/or segment name (see the *FOCUS for IBM Mainframe User's Manual*).

ALIAS

The ALIAS value for each field must either be the full SQL column name (the Interface uses it to generate SQL statements). The ALIAS name must be unique within the segment. DB2 and SQL/DS permit a maximum of 18 alphanumeric characters. The ALIAS name must comply with the same naming conventions described for fieldnames.

USAGE/FORMAT

The USAGE keyword indicates the display format of the field. An acceptable value must include the field type and length and may contain edit options. FOCUS uses the USAGE format for data display on reports and CRTFORMs. All standard FOCUS USAGE formats are available. For a complete list of USAGE formats appropriate for external files, see the *FOCUS for IBM Mainframe User's Manual*.

Note:

- The field *type* described by the USAGE format must be identical to that of the ACTUAL format. For example, a field with an alphanumeric USAGE field type must have an alphanumeric ACTUAL field type.
- Field type text (TX) varies for each RDBMS in terms of syntax and storage limitations. See *ACTUAL Format Conversion Chart* on page 4-7 for a complete discussion.
- Fields with decimal and floating point datatypes must be described with the correct scale and precision. Scale (s) is the number of positions to the right of the decimal point. Precision (p) is the total length of the field. For FOCUS field formats, the field length includes the decimal point and negative sign. RDBMS datatypes exclude positions for the decimal point and negative sign.

  For example, a column defined as DECIMAL(5,2) in DB2 would have a USAGE attribute of P7.2 to allow for the decimal point and a possible negative sign.

ACTUAL Format Conversion Chart

The ACTUAL keyword indicates the FOCUS representation of RDBMS datatypes. The following conversion chart lists common datatypes and their FOCUS representations. Some aspects of ACTUAL = TX or ACTUAL = DATE are unique for the Interface; both formats are discussed after the conversion chart.

SQL Datatype	ACTUAL Format	Description
CHAR(n)	An	Fixed-length alphanumeric, not exceeding 254 characters.
VARCHAR(n)	An	Variable-length character string where n ≤ 254 characters. Trailing blanks are truncated for efficient storage.
VARCHAR(n)	TX	LONG VARCHAR where 254 < n < 4K characters; variable-length text. Trailing blanks are truncated for efficient storage. You may also use TX where n ≤254 characters if you want text wrapping for the field in reports.

SQL Datatype	**ACTUAL Format**	**Description**
SMALLINT	I4	2-byte binary integer, ranges from -32767 to +32768 . **Note:** To create SMALLINT NOT NULL columns with the CREATE FILE command or with the HOLD FORMAT DB2 or HOLD FORMAT SQL facilities, use an ACTUAL attribute of I2 in the CREATE FILE statement, but subsequently change it to I4. For all other purposes, use an ACTUAL attribute of I4.
INTEGER	I4	4-byte binary integer, ranges from -2147483648 to +2147483647.
DECIMAL(p,s)	P((p+1)/2) P8	Packed decimal with format p.s where p excludes the decimal point and s is the scale of the stored value. p must be less than or equal to 31. Required if p≤15 and the column allows nulls (MISSING=ON).
REAL	F4	Single-precision floating-point number (4-byte).
FLOAT	D8	Double-precision floating-point number (8-byte).
DATE	DATE	Standard SQL storage for dates. For information about the default date ('1900-12-31'), see *Default DATE Considerations* in Appendix A, *Additional Topics*. You can use this format with USAGE values containing any combination of year, month, and day.
TIME	A8	Standard SQL storage for time.
TIMESTAMP	A26	Standard SQL storage for timestamps—date and time. The TIMESTAMP value may be updated or inserted if the input string is in the correct DB2 TIMESTAMP format. Consult the IBM DB2 or SQL/DS SQL Reference for the correct input format.
GRAPHIC	Kn	DBCS Kanji character set. Fixed-length string of 'n' 16-bit characters where 0 < n ≤ 127. The appropriate corresponding USAGE format is A(2n+2).

SQL Datatype	ACTUAL Format	Description
VARGRAPHIC	Kn	DBCS Kanji character set. Varying-length string of 'n' 16-bit characters where $0 < n \leq 127$. The appropriate corresponding USAGE format is A(2n+2). **Note:** LONG VARGRAPHIC (or VARGRAPHIC (n) where n>127) is not supported.

ACTUAL = DATE

The Interface fully supports a comprehensive set of date formats. ACTUAL = DATE, in conjunction with USAGE date formats (such as YDM, DMY, MDY), describes columns with DATE datatypes. FOCUS date formats containing any combination of the components year, month, and day can display dates stored with the RDBMS DATE datatype. This feature makes it easier to manipulate dates, and the storage format is compatible with both FOCUS and non-FOCUS programs.

Starting with FOCUS Version 6.8, the default date is '1900-12-31'. For information about setting the default value, see *Default DATE Considerations* in Appendix A, *Additional Topics*.

ACTUAL = DATE makes it possible to:

- Sort by date into date sequence, regardless of the USAGE display format.
- Define date components such as year, month, and day, and extract them from the date field.
- Perform date arithmetic and date comparisons without using special date-handling functions.
- Refer to dates in a natural way (JAN 1 1993, for example) regardless of formats.
- Automatically validate dates in transactions.
- Easily convert dates from one USAGE format to another (YMD to DMY, for example).

Note:

- In report requests, a DATE literal in a WHERE clause must conform to its USAGE format. For example, the field DATE_FLD, with a display format of YMD and an ACTUAL format of DATE, is specified as:

  ```
  WHERE DATE_FLD EQ '930224'
  ```

- When using MODIFY to update a field with ACTUAL = DATE, be aware that if the USAGE format lacks a day and/or month component (YY, for example), the Interface assigns a default day and/or month of '01' to the incoming transaction value (for example, 1993 becomes 19930101).

- RDBMS date fields are century aware. However, if you join an RDBMS table to a non-relational file, use a non-aware date literal, or use a MODIFY transaction file with non-aware dates, you may need to invoke FOCUS Year 2000 solutions such as the DEFCENT and YRTHESH parameters. See your FOCUS documentation for details.

ACTUAL = TX

The ACTUAL = TX field type describes LONG VARCHAR datatypes, that is, columns with VARCHAR(n) where n > 254. Text fields contain variable-length text, usually lengthy descriptions or explanations. The syntax is

```
FIELDNAME=name, ALIAS=sqlcolumn, USAGE=TXnn, ACTUAL=TX,$
```

where:

nn

Defines the length of an output line for display. (Maximum line length is 254 characters.)

You can specify text fields in report and MODIFY requests. SQL restrictions do not permit them to be:

- Indexed.
- Defined as primary key fields.
- Used in joins.

The following are additional limitations on the use of text fields in MODIFY requests:

- MODIFY can reference up to 95 text fields.
- You cannot use text fields in the Scratch Pad Area (HOLD).
- You cannot use text fields in a CRTFORM or with TYPE.
- You can include at most one text field per FIXFORM subcommand; the text field must be the last field in the FIXFORM statement.

In report requests (TABLE), you cannot use text fields:

- In an IF test or WHERE clause.
- As a BY clause or an ACROSS clause.
- In a DEFINE or COMPUTE.

The following HOLD formats do not support text fields:

- DIF
- IFPS
- CALC
- LOTUS

Text fields cannot take advantage of the standard edit options, nor can you use them in a DEFINE. Trailing blanks are truncated for efficient storage.

Limit: FOCUS can handle text fields up to 4K in size. (Text fields greater than 4K are truncated at the 4K boundary.)

For SQL/DS, a LONG VARCHAR column may contain up to 32,767 bytes of text.

For DB2, the maximum physical size of a text value is also 32,767, but the entire row must fit into the database buffer pool. A buffer pool is a main storage area reserved to hold data pages or index pages. The buffer pool size is either 4K or 32K, although 4K is generally assumed. The size of the text field is limited by the total row length and the buffer pool size; therefore, the actual space available may be less than the size of the buffer pool.

For DB2, calculate the maximum size available for TX fields using the formula

```
TXsize= (BPsize - nontext_bytes)
```

where:

BPsize
: Is approximately 4000 bytes for 4K buffers or 32,000 bytes for 32K buffers.

nontext_bytes
: Is the total number of bytes for field types other than text (TX).

To load text field data, use a FIXFORM or PROMPT statement, or use TED from within a MODIFY request. For a complete discussion of text fields, consult the *FOCUS for IBM Mainframe User's Manual*.

MISSING

The Interface supports RDBMS null data. In a table, a null value represents a missing or unknown value; it is not the same as a blank or zero. For example, you can use a column specification that allows nulls for a column that does not have to contain a value in every row (such as a raise amount in a table containing payroll data).

When a FOCUS MODIFY or MAINTAIN procedure enters data into a table, it represents missing data in the table as RDBMS standard null data for columns that allow nulls. At retrieval, null data is translated into the FOCUS missing data display value. The default FOCUS NODATA display value is the period (.).

The MISSING attribute in the Master File indicates whether the RDBMS column accepts null data. When a field declaration omits the MISSING keyword, it defaults to OFF. The syntax is

```
MISSING = {OFF|ON} ,$
```

where:

OFF
: Is the default. In the RDBMS, columns should be created with the NOT NULL attribute (or NOT NULL WITH DEFAULT - DB2 only).

`ON`

FOCUS displays the NODATA value for null data. The column should be created without the NOT NULL attribute.

For null data support:

- The RDBMS table definition must describe a column that allows null data *without* the clause NOT NULL.
- The FOCUS Master File for that table must describe a column that allows null data as a field with the attribute MISSING=ON.

Note: If the column allows null data but the corresponding field in the FOCUS Master File uses the attribute MISSING=OFF, null data appears as a zero or blank. In MODIFY or MAINTAIN, incoming null values for these fields are stored as zero or blank. This practice is not recommended since it can affect the results of SUM or COUNT aggregate operations, as well as allowing the (perhaps unintentional) storage of real values for fields that, in fact, should be null.

Comparing Fields With Null Values

FOCUS and the RDBMS differ slightly in how they compare null field values:

- When two fields contain null values FOCUS considers them equal. The RDBMS considers the result of any comparison between them to be undefined and returns the value FALSE for any such selection condition.
- When one field contains a null value and another field does not, FOCUS considers them *not* equal. The RDBMS considers the result of any comparison between them to be undefined and returns the value FALSE for any such selection condition.

Consider two examples:

```
TABLE FILE X                        TABLE FILE X
PRINT *                             PRINT *
WHERE (FIELD1 EQ FIELD2)            WHERE (FIELD1 NE FIELD2)
END                                 END
```

The following table summarizes the results produced by FOCUS and the RDBMS:

Condition	**Field1 Value**	**Field2 Value**	**FOCUS Result**	**RDBMS Result**
EQ	Null	Null	**True**	**False**
	Null	Not Null	False	False
NE	Null	Null	False	False
	Null	Not Null	**True**	**False**

In most cases, the Interface translates the FOCUS WHERE clause to an SQL WHERE predicate and passes it to the RDBMS for processing. (This translation process is called Interface optimization.) If the Interface does not translate the FOCUS WHERE test to an SQL WHERE predicate, FOCUS applies the selection test and may produce a different answer set.

Because of optimization enhancements introduced in this release, certain requests (outer joins, for example) no longer disable optimization as they did in prior releases. Therefore, the answer set returned may differ from that produced by prior versions of the Interface when FOCUS handled the request. See Chapter 7, *The Interface Optimizer*, for a discussion of optimization.

Optional Field Attributes

The optional field attributes DESCRIPTION, TITLE, DEFINE, and ACCEPT are supported for use with the Interface. Refer to the *FOCUS for IBM Mainframe User's Manual* for information about these attributes.

Note: The Interface does not support the use of FIND in the ACCEPT attribute, nor does it support the keywords GROUP and FIELDTYPE.

Access Files

Each Master File has a corresponding Access File. The name of the Access File (membername in the MVS partitioned data set allocated to DDNAME FOCSQL, or CMS filename with filetype FOCSQL) must be identical to that of the Master File. The Access File associates a segment in the Master File with the table it describes.

The Access File must identify the table and primary key (if there is one). It may also indicate the logical sort order of data and identify storage areas for the table. Access File field declarations can define the precision of packed fields.

For multi-table structures, the Access File also contains KEYFLD and IXFLD keywords that implement embedded JOINs. See *KEYFLD and IXFLD* in Chapter 5, *Multi-Table Structures*, for details.

The following is an Access File for the table EMPINFO:

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1,
 WRITE = YES,  DBSPACE = PUBLIC.SPACE0,$
```

Segment Declarations

The segment declaration in the Access File establishes the link between the FOCUS Master File and the RDBMS table or view. Attributes that constitute the segment declaration are: SEGNAME, TABLENAME, DBSPACE, WRITE, KEYS, and KEYORDER. Values for SEGNAME and TABLENAME are required; the remaining parameters acquire default values if they are omitted.

The syntax for an Access File segment declaration is

```
SEGNAME=segname, TABLENAME= [location.][creator.]table

 [,DBSPACE=storage,] [,WRITE= {YES|NO}] [,KEYS= {0|n}]

  [,KEYORDER=sequence,]  ,$
```

where:

segname

Is the one- to eight-character SEGNAME value from the Master File.

location

Is the DB2 subsystem location name for the Distributed Data Facility; 16 characters maximum. Enclose it in double quotation marks if it begins with a number or special character or contains special characters.

creator

Defaults to the current authorization ID if not specified. Eight characters maximum. Enclose it in double quotation marks if it begins with a number or special character or contains special characters.

table

Is the name of the RDBMS table or view; 18 characters maximum. Enclose it in double quotation marks if it begins with a number or special character or contains special characters.

Note: If any part of the TABLENAME includes a dollar sign ($), enclose that part in double quotation marks, and enclose the *entire* TABLENAME value in single quotation marks.

storage

Is databasename.tablespacename or DATABASE databasename for DB2. Is owner.dbspace for SQL/DS. Enclose it in double quotation marks if it begins with a number or special character or contains special characters.

YES

YES specifies read and write access using MODIFY and MAINTAIN; it is the default.

NO

NO specifies read-only access using FOCUS MODIFY and MAINTAIN.

n

Is a value from 0 to 64; indicates how many columns constitute the primary key. Zero is the default.

`sequence`

Indicates the primary key sort sequence. Valid values are as follows:

`LOW` Indicates ascending primary key sort sequence; LOW is the default.

`ASC` Is a synonym for LOW.

`HIGH` Indicates descending primary key sort sequence.

`DESC` Is a synonym for HIGH.

SEGNAME

The SEGNAME value must be identical to the SEGNAME value in the Master File.

TABLENAME

The TABLENAME attribute identifies the RDBMS table or view. It should contain both the creator ID and the table name. If not specified, the creator defaults to the current authorization ID.

The FOCUS DB2 Interface fully supports the DB2 Distributed Data Facility, including three-part tablenames that incorporate a location identifier (see *The DB2 Distributed Data Facility* in Appendix F, *Additional DB2 Topics*). Tablenames in the Access File may have the following format

```
TABLENAME= [location.][creator.]table ,
```

where:

`location`

Is the DB2 subsystem location name, used if the Distributed Data Facility is in effect; 16 characters maximum. Enclose it in double quotation marks if it begins with a number or special character or contains special characters.

`creator`

Is the authorization ID of the table's creator; eight characters maximum. Enclose it in double quotation marks if it begins with a number or special character or contains special characters.

`table`

Is the name of the RDBMS table or view; 18 characters maximum. Enclose it in double quotation marks if it begins with a number or special character or contains special characters.

The maximum DB2 or SQL/DS length for a fully-qualified tablename is 44. See *The DB2 Distributed Data Facility* in Appendix F, *Additional DB2 Topics*, for a discussion of FOCUS support for the DB2 Distributed Data Facility. All names must conform to the rules for identifiers stated in the appropriate RDBMS manual.

Note: If any part of the TABLENAME contains a dollar sign ($), enclose that part in double quotation marks, and enclose the entire TABLENAME value in single quotation marks. For example:

```
TABLENAME = 'USER1."TABLE$1"'
```

DBSPACE

The DBSPACE keyword identifies the storage area in which the RDBMS table resides. It is used by the FOCUS CREATE FILE command (see *The CREATE FILE Command* in Chapter 6, *Automated Procedures*).

The storage areas identified by the DBSPACE keyword are called tablespaces in DB2 and dbspaces in SQL/DS. The syntax is

```
DBSPACE=storage,
```

where:

`storage`
: Is `databasename.tablespacename` or `DATABASE databasename` for DB2. Is `owner.dbspace` for SQL/DS. Enclose it in double quotation marks if it begins with a number or special character or contains special characters.

In SQL/DS, the RDBMS default value is the SQL/DS authorization ID's private DBSPACE.

In DB2, the RDBMS default value is DSNDB04, a public database. (DB2 automatically generates a tablespace in DSNDB04.)

Note:

- The DBSPACE keyword is ignored for all operations except FOCUS CREATE FILE.
- The Interface may have been installed with a default DBSPACE setting which is used if you do not either specify the DBSPACE attribute or issue a SET DBSPACE command.
- You can also declare storage areas by issuing the SET DBSPACE environmental command prior to CREATE FILE (see *DBSPACE* in Chapter 11, *Environmental Commands*).
- The Access File DBSPACE attribute overrides both the SET command and the installation default.

WRITE

The read/write security parameter, WRITE, determines whether or not the Interface allows FOCUS MODIFY and MAINTAIN operations (INCLUDE, UPDATE, or DELETE) on the table. The syntax is

```
WRITE = {YES|NO}
```

where:

`YES`

: Specifies read and write access using FOCUS MODIFY and MAINTAIN. YES is the default.

`NO`

: Specifies read-only access using FOCUS TABLE, MAINTAIN, and MODIFY. You can use MODIFY or MAINTAIN read-only functions, such as MATCH, NEXT, CRTFORM, or WINFORM, to display rows.

Note:

- The WRITE attribute has no effect on FOCUS reporting operations.
- Regardless of the WRITE value, RDBMS security must approve all operations and activities.
- The Interface must have been installed to allow Read/Write operations. Contact your system support staff for installation options installed at your site.

KEYS

The KEYS attribute indicates how many columns constitute the primary key for the table. Acceptable values range from 0 to 64. Zero, the default, indicates that the table does not have a primary key. In the corresponding Master File, primary key columns must correspond to the first n fields described.

The syntax is

```
KEYS = {0|n} ,
```

where:

n

: Is a value from 0 to 64. Zero is the default.

The KEYS value has the following significance in reporting operations:

- When you use FOCUS FST. or LST. direct operators, the Interface instructs the RDBMS to sort the answer set by the primary key in KEYORDER sequence. (**Note:** LST. processing is automatically invoked if you request SUM or WRITE of an alphanumeric field or use one in a report heading or footing.)
- The Interface uses the KEYS value to determine the relationship between two joined tables. For example, if the primary key of one table is joined to the primary key of another table, the Interface can assume that a one-to-one relationship exists between the two tables. It then uses this assumption in conjunction with the JOIN specification and the current optimization setting to produce SQL statements. See Chapter 7, *The Interface Optimizer*, and Chapter 8, *Advanced Reporting Techniques*, for a detailed explanation.

To provide consistent access to tables, you should specify the KEYS attribute whenever a primary key exists.

The KEYS value also has significance in MODIFY operations (see Chapter 12, *Maintaining Tables With FOCUS*, for a detailed explanation).

KEYORDER

The KEYORDER attribute is optional. It specifies the logical sort sequence of data by the primary key; it does not affect the physical storage of data. The Interface uses the KEYORDER value when you specify FST. and LST. direct operators in report requests. The syntax is

```
KEYORDER= sequence  ,
```

where:

sequence

Indicates the primary key sort sequence. Valid values are as follows:

LOW	Sorts the rows in ascending primary key sequence. LOW is the default.
ASC	Is a synonym for LOW.
HIGH	Sorts the rows in descending primary key sequence.
DESC	Is a synonym for HIGH.

For example, to retrieve the most recent pay dates first, specify KEYORDER = HIGH for the SALINFO table:

```
SEGNAME = SALINFO, TABLENAME = "USER1"."SALINFO", KEYS = 2,
 WRITE = YES,  KEYORDER = HIGH, DBSPACE = PUBLIC.SPACE0,$
```

The Interface requests rows ordered by SALEID and PAY_DATE in descending order. FST.PAY_DATE retrieves the most recent salary data for each employee. See the Master File for SALINFO in Appendix C, *File Descriptions and Tables*.

KEYORDER also determines the logical sort order for MODIFY and MAINTAIN NEXT subcommands (see Chapter 12, *Maintaining Tables With FOCUS*).

Field Declarations

Access File field declarations define the precision type for DECIMAL fields.

PRECISION

Static TABLE requests can contain SQL host variables (see *SQL COMPILE and SQL RUN Processing* in Chapter 13, *Static SQL*). When you use these variables in an SQL predicate, the RDBMS optimizer considers them for indexed access only if they have the same lengths as the corresponding RDBMS columns.

The ACTUAL attribute from the Master File does not provide enough information for determining the RDBMS column length of DECIMAL columns. For example, an ACTUAL attribute of P6 can correspond to a DECIMAL column of precision 10 or 11.

The PRECISION attribute in the Access File defines the precision for DECIMAL columns. The syntax is

```
FIELD = fieldname, PRECISION = {ODD|EVEN} , $
```

where:

fieldname

Is the name of an indexed packed decimal field that will be used either in screening conditions or as a join field in a cross-referenced file in a FOCUS-managed join.

`ODD|EVEN`

Is the precision-type. If n is the length specified for the ACTUAL attribute in the Master File:

`ODD`, the default, calculates the DECIMAL column precision for the host variable using the formula 2n-1.

`EVEN` uses the formula 2n-2.

For example, an ACTUAL of P6 and a precision-type of ODD (the default) makes the host variable length 11; the RDBMS can consider index access if the RDBMS column precision is 11 (provided the column is indexed). With precision-type EVEN, the host variable length is 10, which enables index access for DECIMAL columns of precision 10.

Note: The SQL/DS optimizer in Version 3 Release 4 and below does not allow index access from static SQL for packed decimal columns of even precision, regardless of the precision-type used.

The FOCUS OCCURS Segment

For use with FOCUS TABLE requests, you can describe tables that contain repeating columns as FOCUS OCCURS segments. Repeating data is not characteristic of normalized tables and views; however, denormalized tables may exist for some situations.

The FOCUS OCCURS segment, a virtual construct, eliminates the need to specify the names of every column in a TABLE request if a group of columns contains similar data. In the OCCURS segment definition, you redefine all the separate columns as one group that shares the same name. You can then define the order field, an internal FOCUS counter that enables you to access specific columns by their sequence numbers within the group instead of by separate names.

To define a FOCUS OCCURS segment, you need:

- An additional segment declaration in the Master File to define the OCCURS segment.
- One field declaration to represent the repeating fields.
- Optionally, a field declaration for the ORDER field, a counter that is internal to FOCUS and not contained in the RDBMS.

Creating an OCCURS Segment

To create an OCCURS segment:

1. Describe the entire table as a single-table Master File. Include a field declaration for each repeating column.
2. Add a segment description for the related OCCURS segment. In it, include a field declaration that redefines the repeating portion of the table.

The syntax for an OCCURS segment declaration is

```
SEGNAME=segname, PARENT=name, POSITION=field, OCCURS=nnnn,$
```

where:

segname
: Is the name of the OCCURS segment, up to eight characters.

parent
: Is the SEGNAME value of the table that contains declarations for the repeating fields.

field
: Is the name of the first repeating field in the parent table.

nnnn
: Is the number of repeating fields. Acceptable values range from 1 to 4095.

Note:

- The OCCURS segment declaration in the Master File must not include the SEGTYPE keyword.
- The Access File must not contain a corresponding segment declaration for the OCCURS segment.
- OCCURS segments are available for TABLE operations. To change data, you must specify the individual repeating fields from the parent segment.
- Using an OCCURS segment disables Interface optimization.

The following SALARY table contains monthly payroll tax deductions for an employee, and the Master File describes the repeating columns as 12 separate deduction fields. It also includes an OCCURS segment, OCC:

```
FILENAME=SALARY, SUFFIX=SQLDS,$

SEGNAME=SALARY,  SEGTYPE=S0,$
 FIELD=EMPID,    ALIAS=EMPID,    USAGE=A7,   ACTUAL=A7,$
 FIELD=EMPNAME,  ALIAS=EMPNAME,  USAGE=A10,  ACTUAL=A10,$
 FIELD=SALARY,   ALIAS=PAY,      USAGE=P9.2, ACTUAL=P4,$

 FIELD=DEDUCT1,  ALIAS=DEDUCT1,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT2,  ALIAS=DEDUCT2,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT3,  ALIAS=DEDUCT3,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT4,  ALIAS=DEDUCT4,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT5,  ALIAS=DEDUCT5,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT6,  ALIAS=DEDUCT6,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT7,  ALIAS=DEDUCT7,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT8,  ALIAS=DEDUCT8,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT9,  ALIAS=DEDUCT9,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT10, ALIAS=DEDUCT10, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT11, ALIAS=DEDUCT11, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT12, ALIAS=DEDUCT12, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$

SEGNAME=OCC, PARENT=SALARY, POSITION=DEDUCT1, OCCURS=12,$
 FIELD=TAX,      ALIAS=TAXDEDUC, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
```

The OCCURS segment redefines the 12 deduction fields in the SALARY segment, beginning with DEDUCT1. The TAX field in the OCCURS segment represents the 12 repeating fields.

The corresponding Access File does not contain a declaration for the OCCURS segment:

```
SEGNAME = SALARY, TABLENAME = "USER1"."SALARY", KEYS = 1,
WRITE = NO,   DBSPACE = PUBLIC.SPACE0,$
```

The ORDER Field

The ORDER field is a FOCUS counter that assigns a sequence number to each field within a group of repeating fields. Specify this optional field when the order of data is significant. The ORDER field does not represent an existing column; it is used only for internal processing.

The ORDER field must be the last field described in the OCCURS segment. The syntax is

```
FIELD=name, ALIAS=ORDER, USAGE=In, ACTUAL=I4,$
```

where:

name
: Is any meaningful name.

I*n*
: Is an integer (In) format.

Note:

- The value of ALIAS must be ORDER.
- The value of ACTUAL must be I4.

In the previous SALARY table example, no column explicitly specifies the month for each TAX field. To associate the month, the next example adds the ORDER field as the last field in the OCCURS segment:

```
FILENAME=SALARY, SUFFIX=SQLDS,$

SEGNAME=SALARY,  SEGTYPE=S0,$
 FIELD=EMPID,    ALIAS=EMPID,    USAGE=A7,   ACTUAL=A7,$
 FIELD=EMPNAME,  ALIAS=EMPNAME,  USAGE=A10,  ACTUAL=A10,$
 FIELD=SALARY,   ALIAS=PAY,      USAGE=P9.2, ACTUAL=P4,$

 FIELD=DEDUCT1,  ALIAS=DEDUCT1,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT2,  ALIAS=DEDUCT2,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT3,  ALIAS=DEDUCT3,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT4,  ALIAS=DEDUCT4,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT5,  ALIAS=DEDUCT5,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT6,  ALIAS=DEDUCT6,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT7,  ALIAS=DEDUCT7,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT8,  ALIAS=DEDUCT8,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT9,  ALIAS=DEDUCT9,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT10, ALIAS=DEDUCT10, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT11, ALIAS=DEDUCT11, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT12, ALIAS=DEDUCT12, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$

SEGNAME=OCC, PARENT=SALARY, POSITION=DEDUCT1, OCCURS=12,$
 FIELD=TAX,      ALIAS=TAXDEDUC, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=ORDER,    ALIAS=ORDER,    USAGE=I4,   ACTUAL=I4,$
```

In subsequent report requests, you can use the DECODE function to translate the ORDER field into monthly values.

In this example, a DEFINE statement assigns the month to each counter value. You can specify a temporary field before the report request or in the Master File:

```
> > define file salary
> month/a3=decode order(1 'jan' 2 'feb' 3 'mar' 4 'apr' 5 'may'
> 6 'jun' 7 'jul' 8 'aug' 9 'sep' 10 'oct' 11 'nov' 12 'dec' else '   ');
> end
> > table file salary
> print last_name tot.tax if month eq 'jan'
> end
  NUMBER OF RECORDS IN TABLE=     5  LINES=    5
```

You can also use the ORDER field in selection tests. For example:

```
> > table file salary
> print empeid last_name tax
> if order eq 12
> end
  NUMBER OF RECORDS IN TABLE=     5  LINES=    5
```

5 Multi-table Structures

Topics:

- Advantages of Multi-table Structures
- Creating a Multi-table Structure
- Multi-field Embedded Join

With the Interface, you can describe two or more related tables in a single Master and Access File pair. This type of multi-table structure is called an *embedded join*.

Within a multi-table structure, each participating table must have at least one field in common with at least one other table in the structure. Typically, this common field is the primary key of one table and the foreign key of the other. A single Master and Access File pair can relate up to 64 separate tables in this manner.

Multi-table Master and Access Files describe the relationships between tables. The Interface implements these relationships at run time by matching values in fields common to two or more tables. A report or maintenance procedure can refer to any or all of the tables included in the multi-table description.

In this chapter, the terms *primary key* and *foreign key* refer to the common fields in two related tables. These may or may not have been described as primary and foreign keys in SQL CREATE TABLE statements (RDBMS referential integrity). In practice, FOCUS can use any two fields that share a common format to relate tables in multi-table file descriptions.

Note: This chapter describes manual methods for creating Master File and Access Files. Chapter 6, *Automated Procedures,* describes an automated method for creating Master and Access Files.

Advantages of Multi-table Structures

The following are advantages of defining multiple tables in a single FOCUS Master File:

- For reporting, a multi-table structure automatically joins tables referenced in the report request, so relationships between tables can be pre-defined for a user without creating an additional RDBMS view.
- A multi-table structure creates a FOCUS logical view of the data tailored to contain only those columns that should be seen by a user. The FOCUS DBA security feature can define additional levels of security.

- Tables described in a multi-table Master File can be maintained together using the FOCUS MODIFY and MAINTAIN facilities and can take advantage of FOCUS referential integrity provided by the Interface. (Refer to Chapter 12, *Maintaining Tables With FOCUS*, for a description of MODIFY, MAINTAIN, and FOCUS referential integrity.)
- TABLE requests access only those RDBMS tables that contain columns referenced (either explicitly or implicitly) in the request.

  A FOCUS TABLE request that references a dynamically joined structure generates SQL join predicates for all segments in the subtree that starts from the root segment. Multi-table Master Files do not necessarily generate these predicates; in a multi-table structure, the subtree effectively begins with the highest referenced segment. This difference may cause identical TABLE requests to produce different reports when run against a dynamic join structure and a multi-table Master File that represent the same tree structure. See *The Dynamic JOIN Command* in Chapter 8, *Advanced Reporting Techniques,* for a discussion of dynamic joins.

Creating a Multi-table Structure

To create a multi-table structure, describe the tables, and the relationships between them, in a single Master and Access File.

Multi-table Master Files

All segment declarations in a multi-table Master File must describe RDBMS tables or views. Each segment represents one table or view, up to a total of 64 segments. An RDBMS view counts as one segment toward the total, even if the view represents a join of two or more tables.

If several Master Files (used only with TABLE requests) include the same table, you can avoid repeating the same description multiple times. Describe the table in *one* of the Master Files, and use the CRFILE attribute in the other Master Files to access the existing description. For a full explanation of remote segment descriptions, see *Remote Segment Descriptions* in Appendix A, *Additional Topics*.

The embedded join facility uses existing FOCUS Master File syntax (specifically the PARENT segment attribute) to describe the relationships between tables

```
FILENAME=mtname, SUFFIX=SQLDS   [,$]

 SEGNAME=table1, SEGTYPE= {S0|KL} [,CRFILE=crfile1] [,$]

  FIELD=name,...,$
  .
  .
  .
 SEGNAME=table2, SEGTYPE=relationship, PARENT=table1 [,CRFILE=crfile2] [,$]

  FIELD=name,...,$
  .
  .
  .
```

where:

mtname
: Is the one- to eight-character name of the multi-table Master File.

table1
: Is the SEGNAME value for the parent table. If this segment references a remote segment description, table1 must be identical to the SEGNAME from the Master File that contains the full definition of the RDBMS table's columns (see *Remote Segment Descriptions* in Appendix A, *Additional Topics*).

name
: Is any fieldname.

table2
: Is the SEGNAME value for the related table. If this segment references a remote segment description, table2 must be identical to the SEGNAME from the Master File that contains the full definition of the RDBMS table's columns.

relationship
: Indicates the type of relationship between the table and its parent. Valid values are as follows:

`S0` indicates that the related table is in a one-to-many or many-to-many (non-unique) relationship with the table named as its parent.

`U` indicates that the related table is in a one-to-one or a many-to-one (unique) relationship with the table named as its parent.

`KL` references a remote segment description. Indicates that the related table is in a one-to-many or many-to-many (non-unique) relationship with the table named as its parent.

`KLU` references a remote segment description. Indicates that the related table is in a one-to-one or a many-to-one (unique) relationship with the table named as its parent.

`crfile1`

References a remote segment description. Indicates the name of the Master File that contains the full definition of table1’s columns.

`crfile2`

References a remote segment description. Indicates the name of the Master File that contains the full definition of table2’s columns.

SEGNAME

The SEGNAME attribute names the segment. SEGNAME values must be unique within the Master File. If the segment references a remote Master File, its SEGNAME value must be identical to the SEGNAME from the Master File that contains the full definition of the RDBMS table's columns.

SEGTYPE

The SEGTYPE attribute indicates how a table participates in a relationship. The SEGTYPE of the first segment in a multi-table Master File is always S0 (or KL for a remote segment description). Thereafter, related tables (additional segments) are described as follows:

- SEGTYPE=S0 (zero) indicates that the related table is in a one-to-many or many-to-many (non-unique) relationship with the table named as its parent. (For every row of the parent table there may be more than one matching row in the related table.)
- SEGTYPE=U indicates that the related table is in a one-to-one or a many-to-one (unique) relationship with the table named as its parent. (For every row of the parent table, there is at most one matching row in the related table. For a one-to-one relationship to exist, both tables must share the same primary key. For a many-to-one relationship to exist, the primary key of the related table must be a subset of the primary key of the parent table.)
- SEGTYPE=KL references a remote segment description. It indicates that the related table is in a one-to-many or many-to-many (non-unique) relationship with the table named as its parent. (For every row of the parent table there may be more than one matching row in the related table.)
- SEGTYPE=KLU references a remote segment description. It indicates that the related table is in a one-to-one or a many-to-one (unique) relationship with the table named as its parent. (For every row of the parent table, there is at most one matching row in the related table. For a one-to-one relationship to exist, both tables must share the same primary key. For a many-to-one relationship to exist, the primary key of the related table must be a subset of the primary key of the parent table.)

PARENT

All segment declarations other than the first require the PARENT attribute. The PARENT value for a segment is the SEGNAME of the table to which it will be related at run time.

CRFILE

Specify the CRFILE attribute only if the actual description of the table’s columns is stored in another (remote) Master File. The CRFILE value must be the name of the Master File that contains the full definition of the RDBMS table's columns. For a complete discussion of remote segment descriptions, see *Remote Segment Descriptions* in Appendix A, *Additional Topics*.

FIELD

FOCUS fieldnames can consist of up to 66 alphanumeric characters (including any filename and segname qualifiers and qualification characters you may later prefix to them in requests). Within the Master File, fieldnames cannot include qualifiers. Column names are acceptable values if they meet the following FOCUS naming conventions:

- A name can consist of letters, digits, and underscore characters. Special characters and embedded blanks are not advised.
- The name must contain at least one letter.

Since the fieldname appears as the default column title for reports, select a name that is representative of the data. In TABLE, GRAPH, and MODIFY requests, you can specify fieldnames, aliases, or a unique truncation of either. MAINTAIN does not support alias names or truncated names. In all requests, you can qualify a fieldname with its filename and/or segment name.

Fieldnames must be unique within a single segment. If fieldnames are duplicated across segments, use the segment name as a high level qualifier when referencing them in requests.

ALIAS

The ALIAS value for each field must be the full SQL column name (the Interface uses it to generate SQL statements). The ALIAS name must be unique within the segment. DB2 and SQL/DS permit a maximum of 18 alphanumeric characters. The ALIAS name must comply with the same naming conventions described for fieldnames.

ALIAS names may be duplicated within the Master File if they are defined for different tables.

ECOURSE Master File

The following Master File relates the EMPINFO and COURSE tables. The EMPINFO table is described first; therefore, its segment declaration does not include the PARENT attribute. In the COURSE segment declaration, the PARENT keyword identifies EMPINFO as the parent and notifies the Interface that the two tables may be joined at run time for reporting purposes. The FILENAME ECOURSE identifies this relationship. (For an example of a multi-table Master File with a remote segment, refer to *Remote Segment Descriptions* in Appendix A, *Additional Topics*):

```
FILENAME=ECOURSE     ,SUFFIX=SQLDS, $

SEGNAME=EMPINFO      ,SEGTYPE=S0, $
 FIELD=EMP_ID        ,ALIAS=EID         ,USAGE=A9    ,ACTUAL=A9,$
 FIELD=LAST_NAME     ,ALIAS=LN          ,USAGE=A15   ,ACTUAL=A15,$
 FIELD=FIRST_NAME    ,ALIAS=FN          ,USAGE=A10   ,ACTUAL=A10,$
 FIELD=HIRE_DATE     ,ALIAS=HDT         ,USAGE=YMD   ,ACTUAL=DATE,$
 FIELD=DEPARTMENT    ,ALIAS=DPT         ,USAGE=A10   ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY,ALIAS=CSAL        ,USAGE=P9.2  ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE  ,ALIAS=CJC         ,USAGE=A3    ,ACTUAL=A3,$
 FIELD=ED_HRS        ,ALIAS=OJT         ,USAGE=F6.2  ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN    ,ALIAS=BONUS_PLAN  ,USAGE=I4    ,ACTUAL=I4,$

SEGNAME=COURSE ,SEGTYPE=S0  ,PARENT=EMPINFO,$
 FIELD=CNAME   ,ALIAS=COURSE_NAME ,USAGE=A15, ACTUAL=A15,$
 FIELD=WHO     ,ALIAS=EMP_NO      ,USAGE=A9,  ACTUAL=A9,$
 FIELD=GRADE   ,ALIAS=GRADE       ,USAGE=A1,  ACTUAL=A1, MISSING=ON,$
 FIELD=YR_TAKEN,ALIAS=YR_TAKEN    ,USAGE=A2,  ACTUAL=A2,$
 FIELD=QTR     ,ALIAS=QUARTER     ,USAGE=A1,  ACTUAL=A1,$
```

Note: Multi-table Master Files used in MODIFY and MAINTAIN procedures invoke FOCUS referential integrity. Refer to *FOCUS Referential Integrity* in Chapter 12, *Maintaining Tables With FOCUS*, for a discussion of referential integrity before deciding whether to use the same file descriptions for both reporting and maintenance procedures.

Multi-table Access Files

A multi-table Master File indicates that tables are related. However, to implement a join (TABLE) or FOCUS referential integrity (MODIFY and MAINTAIN), you must identify their common fields to FOCUS in the corresponding Access File.

The multi-table Access File includes a segment declaration for each table described in the Master File, even if the segment was referenced remotely in the Master File. The order of segment declarations in the Access File does not have to match the order in the Master File, but maintaining the same order enhances readability.

Each segment declaration in the Access File must contain the required keywords described in *Access Files* in Chapter 4, *Describing Tables to FOCUS*.

In addition, each segment except the first must identify the fields that it shares with its parent segment. In each Access File segment declaration other than the first, the KEYFLD and IXFLD attributes supply the names of the primary key and foreign key fields that implement the relationships established by the multi-table Master File.

The syntax for a multi-table Access File is

```
SEGNAME=table1, TABLENAME=tname1,...,$
SEGNAME=table2, TABLENAME=tname2,...,
  KEYFLD=pkfield, IXFLD=fkfield,$
```

where:

table1
: Is the SEGNAME of the parent table from the multi-table Master File.

table2
: Is the SEGNAME of the related table from the multi-table Master File.

tname1
: Is either `tablename` or `creator.tablename` for the parent table. For the DB2 Distributed Data Facility, also include a subsystem location identifier (see *The DB2 Distributed Data Facility* in Appendix F, *Additional DB2 Topics*).

tname2
: Is either `tablename` or `creator.tablename` for the related table. For the DB2 Distributed Data Facility, also include a subsystem location identifier (see *The DB2 Distributed Data Facility* in Appendix F, *Additional DB2 Topics*).

pkfield
: Is the fieldname of the primary key column in the parent (table1) table.

fkfield
: Is the fieldname of the foreign key column in the related (table2) table.

KEYFLD and IXFLD

The KEYFLD and IXFLD keywords identify the field shared by a related table pair. KEYFLD is the FIELDNAME of the common column from the parent table. IXFLD is the FIELDNAME of the common column from the related table. KEYFLD and IXFLD must have the same datatype. It is recommended, but not required, that their lengths also be the same.

Note: An RDBMS index on both the KEYFLD and IXFLD columns provides the RDBMS with a greater opportunity to produce efficient joins. The columns must have the same datatype. If their length is the same, the RDBMS handles the join more efficiently.

In the ECOURSE example from *ECOURSE Master File* on page 5-6, the fields EMP_ID in EMPINFO and WHO in COURSE both contain employee identification numbers; they represent the common field. Since the COURSE table is the related table, its segment declaration in the ECOURSE Access File identifies these common columns from EMPINFO and COURSE:

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1,
   WRITE = YES,  DBSPACE = PUBLIC.SPACE0,$
SEGNAME = COURSE, TABLENAME = "USER1"."COURSE", KEYS = 2,
   WRITE = YES,     DBSPACE = PUBLIC.SPACE0,
     KEYFLD = EMP_ID, IXFLD = WHO,$
```

Multi-field Embedded Join

In relational systems, a relationship or link between tables can depend on multiple columns. Embedded joins defined in multi-table Master and Access Files also provide this ability. (The dynamic JOIN command supports this feature as well; see *The Dynamic JOIN Command* in Chapter 8, *Advanced Reporting Techniques*).

To describe a multi-field join for a multi-table structure, specify multiple fieldnames for the KEYFLD and IXFLD attributes in the Access File. Separate the component fields participating in the multi-field join with slash (/) symbols.

The syntax for a multi-table Access File with a multi-field link is

```
SEGNAME=name1, TABLENAME=table1,...,$
SEGNAME=name2, TABLENAME=table2,...,
   KEYFLD=pkfield1/pkfield2,
   IXFLD=fkfield1/fkfield2,$
```

where:

name1
: Is the SEGNAME of the parent table from the multi-table Master File.

name2
: Is the SEGNAME of the related table from the multi-table Master File.

table1
: Is either *tablename* or *creator.tablename* for the parent table. For the DB2 Distributed Data Facility, also include a subsystem location identifier (see *The DB2 Distributed Data Facility* in Appendix F, *Additional DB2 Topics*).

table2
: Is either *tablename* or *creator.tablename* for the related table. For the DB2 Distributed Data Facility, also include a subsystem location identifier (see *The DB2 Distributed Data Facility* in Appendix F, *Additional DB2 Topics*).

pkfield1/pkfield2
: Are the fieldnames that compose the primary key in the parent (or host) table.

`fkfield1/fkfield2`

Are the fieldnames that compose the foreign key in the related table.

Up to 16 fields can participate in a link between two tables. The fields that constitute this multi-field relationship do not have to be contiguous either within the table or the FOCUS Master File.

The number and order of fields for the KEYFLD value must correspond to those for the IXFLD value.

If the list of fields exceeds one line (80 characters), continue it on a second line. You can use as many lines as necessary, provided that each line is filled up to and including the 80th position. The 80th position cannot contain a slash (/) character.

To illustrate the multi-field join, suppose that the fields LAST_NAME and FIRST_NAME compose the primary key for the EMPINFO1 table. Also, assume that fields the LNAME and FNAME serve as the common fields (foreign key) in the COURSE1 table.

The ECOURSE1 Master File (defined solely for the purpose of this example and not included in Appendix C, *File Descriptions and Tables*) reflects the new fields:

```
FILENAME=ECOURSE1         ,SUFFIX=SQLDS,$

SEGNAME=EMPINFO1          ,SEGTYPE=S0,$
 FIELDNAME=LAST_NAME      ,ALIAS=LN              ,USAGE=A15  ,ACTUAL=A15,$
 FIELDNAME=FIRST_NAME     ,ALIAS=FN              ,USAGE=A10  ,ACTUAL=A10,$
 FIELDNAME=HIRE_DATE      ,ALIAS=HDT             ,USAGE=YMD  ,ACTUAL=DATE,$
 FIELDNAME=DEPARTMENT_CD  ,ALIAS=DEPARTMENT_CD   ,USAGE=A10  ,ACTUAL=A10,
   MISSING=ON ,$
 FIELDNAME=CURRENT_SALARY ,ALIAS=CURRENT_SALARY ,USAGE=P9.2 ,ACTUAL=P4, $
 FIELDNAME=CURR_JOBCODE   ,ALIAS=CJC             ,USAGE=A3   ,ACTUAL=A3,$
 FIELDNAME=ED_HRS         ,ALIAS=OJT             ,USAGE=F6.2 ,ACTUAL=F4,
   MISSING=ON ,$
 FIELDNAME=BONUS_PLAN     ,ALIAS=BONUS_PLAN      ,USAGE=I4   ,ACTUAL=I4,$

SEGNAME=COURSE1           ,SEGTYPE=S0            ,PARENT=EMPINFO1, $
 FIELD=CNAME              ,ALIAS=COURSE_NAME     ,USAGE=A15  ,ACTUAL=A15,$
 FIELD=LNAME              ,ALIAS=LNAME           ,USAGE=A15  ,ACTUAL=A15,$
 FIELD=FNAME              ,ALIAS=FNAME           ,USAGE=A10  ,ACTUAL=A10,$
 FIELD=GRADE              ,ALIAS=GRADE           ,USAGE=A1   ,ACTUAL=A1,
   MISSING=ON ,$
 FIELD=YR_TAKEN           ,ALIAS=YR_TAKEN        ,USAGE=A2   ,ACTUAL=A2, $
 FIELD=QTR                ,ALIAS=QUARTER         ,USAGE=A1   ,ACTUAL=A1, $
```

In the ECOURSE1 Access File, the KEYFLD and IXFLD values consist of fieldnames separated by a slash (/) :

```
SEGNAME = EMPINFO1 ,TABLENAME = "USER1"."EMPINFO1" ,KEYS = 2,WRITE = YES,
   DBSPACE = PUBLIC.SPACE0,$
SEGNAME = COURSE1  ,TABLENAME = "USER1"."COURSE1"  ,KEYS = 3,WRITE = YES,
   DBSPACE = PUBLIC.SPACE0,
      KEYFLD=LAST_NAME/FIRST_NAME,IXFLD=LNAME/FNAME, $
```

6 Automated Procedures

Topics:

- Creating File Descriptions
- The CREATE FILE Command

This chapter describes the following FOCUS procedures:

- The AUTODB2 facility for DB2 creates FOCUS Master and Access Files for existing DB2 tables.
- The AUTOSQL facility for SQL/DS creates FOCUS Master and Access Files for existing SQL/DS tables.
- The FOCUS CREATE FILE command creates a DB2 or SQL/DS table from the description in an existing FOCUS Master and Access File.

The FOCUS AUTODB2 and AUTOSQL facilities use RDBMS catalog definitions of existing tables or views, along with user selections, to generate Master and Access Files (see *Creating File Descriptions* on page 6-1). The AUTODB2 and AUTOSQL facilities work with any RDBMS release.

Note: Do not use a Master File created (by AUTODB2 or AUTOSQL) in one FOCUS release together with an Access File created in a different FOCUS release.

The FOCUS CREATE FILE command uses existing FOCUS Master and Access Files to generate an RDBMS table definition. CREATE FILE also generates a unique index if the KEYS value in the Access File is greater than zero (see *The CREATE FILE Command* on page 6-28).

Creating File Descriptions

The AUTODB2 and AUTOSQL facilities are FOCEXECs that gather column information from RDBMS catalog tables and use the information to create Master and Access Files.

When you invoke the AUTODB2 and AUTOSQL facilities, they present you with a series of menus that prompt you to identify the tables you want to describe and the relationships between them. If you are unfamiliar with any of the terms used in this section, review Chapter 5, *Multi-Table Structures*. Subsequent sections explain each menu in detail, and *AUTODB2 Sample Session* on page 6-19 contains a sample session.

The following is a brief overview of the AUTODB2 and AUTOSQL screens:

- The initial screen, or Main Menu, allows you to name your Master and Access Files and to identify the tables that may contain columns needed for defining them. You can use wildcard characters to generate a list of tables from which to choose. You can also change certain default values that are displayed on this screen.
- The Table Selection Screen displays the list of tables generated as a result of your entries on the Main Menu. You can select tables from the list displayed. If you choose more than one table, you must designate one to be the root and the rest to be children.

  Note: Descendant is a synonym for child. Both terms, used interchangeably in the following discussion, refer to related tables, as described in Chapter 5, *Multi-table Structures*.

  If you select just a root table, and no children, on the Table Selection Screen, AUTODB2 or AUTOSQL creates your single-table Master and Access File at this point. If you choose at least one child table, you proceed to the Child Selection Screen.
- The Child Selection Screen prompts you to identify pairs of related tables.
- The Common Column Selection Screen asks you to identify the primary and foreign key columns for each pair of related tables (see Chapter 5, *Multi-table Structures*, for an explanation of multi-table structures).

After you describe all the relationships, AUTODB2 or AUTOSQL creates the Master and Access Files. *How to Use AUTODB2 and AUTOSQL* on page 6-3 contains sample screens and explains AUTODB2 and AUTOSQL defaults.

You can use the resulting pair of file descriptions immediately or edit them to include such options as DEFINE statements. As with any Master File, you can add additional security by including FOCUS DBA security attributes.

Note: If a table has a Master File created in a prior FOCUS release, and if you re-run AUTODB2 or AUTOSQL on the table, some fieldnames and/or USAGE formats generated in the new Master File may differ from those in the old Master File. As a result, requests that ran against the old Master File, and DEFINE fields based on field length, may require changes.

The RDBMS system catalog table, SYSCOLUMNS, provides such column information as column names, datatypes, column lengths, and whether null values are allowed. System catalog tables SYSKEYS (DB2) and SYSINDEXES provide unique index information. The RDBMS searches for the first unique index created and uses the columns on which this index is defined as the primary key.

In addition to the ability to create multi-table Master and Access Files in interactive mode (batch mode remains limited to one table description per execution), the Interface AUTODB2 and AUTOSQL facilities include the following three key features:

- They provide an option on the Main Menu for assigning a USAGE attribute of either x or x.0 to table columns of data type DECIMAL(x,0).
- They provide an option on the Main Menu for changing default Main Menu entries and logging the new default values to a text file for repeated use.
- They provide an extensive on-line help facility.

AUTODB2 uses only FOCUS DYNAM dynamic allocation, freeing it from any dependence on TSO. The MSO run-time requirements and user interface are identical to those of AUTODB2 running in the TSO environment; the user or installer need make no modifications. Users familiar with AUTODB2 in the TSO environment will find it identical under MSO.

You can execute AUTODB2 and AUTOSQL in batch mode for single-table structures by supplying the necessary execution parameters at invocation, eliminating the need for input screens. For AUTODB2, batch mode execution is possible in the MSO and TSO environments as well as in MVS batch. *AUTODB2 and AUTOSQL in Batch Mode* on page 6-15 explains batch execution.

How to Use AUTODB2 and AUTOSQL

Before starting AUTODB2 or AUTOSQL, it is helpful to know the names of the tables or views you will be using (including, if possible, their creator names and, for DB2, their database names). Having this information in advance can avert a costly search of the RDBMS catalogs using wildcard characters from the Main Menu. Get this information from your RDBMS database administrator, or query the system table SYSCOLUMNS using Master File DB2CAT for DB2 or SYSCOL for SQL/DS. (See *Querying the RDBMS SYSCOLUMNS Catalog* in Appendix A, *Additional Topics*, for a sample request.)

The first time you execute AUTODB2 or AUTOSQL, many of the entry fields on the Main Menu screen display default values. You can customize the default values for your application and use PF4 to store them in a parameter log file for future use. The PFkey functions are explained following the description of the Main Menu entry fields.

You also need enough available disk space. The AUTOSQL facility writes the new file descriptions directly to the disk accessed as filemode A. AUTODB2 writes the file descriptions directly to the data sets specified on the Main Menu.

To start AUTODB2 or AUTOSQL, enter the FOCUS environment and type the following from the FOCUS command level:

```
EX {AUTODB2|AUTOSQL}
```

Press the Enter key.

The Main Menu

The following is an example of the Main Menu for AUTODB2. AUTOSQL displays a similar screen (without the entry fields for the target data set files). Complete the entry fields on the Main Menu and press Enter to begin processing:

```
Main Menu     Master File Generation Facility for DB2
      Master Filename ================> autoemp

      Information extracted from:
        Creator Name (or *) ==========> user1
        Table Name   (or *) ==========> *
        Database Name ================> *
      Description will be a member of:
        Master Target PDS => USER1.MASTER.DATA
        Access Target PDS => USER1.FOCSQL.DATA
        FOCDEF Target PDS => USER1.FOCDEF.DATA

        Replace Existing Description?=> Y       (Y/N)
        Read/Write Functionality =====> W       (R=Read,W=Write)
        Date Display Format ==========> YYMD
        Display Decimal when SCALE=0?=> Y       (Y/N)
        Use LABEL as Column Heading? => N       (Y/N)
        Use Remarks for FOCDEF? ======> N       (Y/N)
        Use Creator Name in AFD? =====> Y       (Y/N)
        Use Long Fieldnames? =========> Y       (Y/N)
        Parm File => USER1.FOCSQL.DATA

PF1=Help PF2=Restart PF3=Exit PF4=Log PF5=MFD PF6=AFD PF9=Picture PF10=List
```

The following list describes the Main Menu entry fields:

Master Filename	Is the 1- to 8-character name you select for referring to the data in requests. In MVS, this name must be a valid member name. In CMS, this name must be a valid filename.

Creator Name (or *)	Is the 1- to 8-character creator name of the tables that you want to describe. Specify an asterisk (*) to select tables for all creators. The default is your userid. You can use the asterisk (*) wildcard character in the creator, table, and database name entry fields to create a list based on the provided pattern. **Note:** You cannot enter a name that contains a dollar sign ($) in the creator, table name, or database name entry fields; however, with the asterisk (*) wildcard character, you can generate a list that includes names containing dollar signs. You can then choose tables from this list.
Table Name (or *)	Is the 1- to 18-character name of an existing table that you want to describe. Specify an asterisk (*), the default, to select all tables.
Database Name (or *)	Applies to MVS only. Is the 1- to 8-character name of the database that contains any or all tables you may select. Specify an asterisk (*), the default, to select all databases. You can omit this value if you provide creator or table.
Master Target PDS	Applies to MVS only. Is the fully qualified data set name of the Master File PDS in which to store the Master File. Do not use quotation marks (single or double) in the data set name. The default is 'userid.MASTER.DATA'.
Access Target PDS	Applies to MVS only. Is the fully qualified data set name of the Access File PDS in which to store the Access File. Do not use quotation marks (single or double) in the data set name. The default is 'userid.FOCSQL.DATA'.
FOCDEF Target PDS	Applies to MVS only and is required only if you specify Yes (Y) in the “Use REMARKS for FOCDEF?” field. Is the fully qualified data set name of the FOCDEF File PDS in which to store the TableTalk® Help file. Do not use quotation marks (single or double) in the data set name. The default is 'userid.FOCDEF.DATA'.
Replace Existing Description?	Specifies whether to overwrite existing Master and Access Files (Y/N). No (N) is the default. Yes (Y) replaces existing descriptions of the same name.

Read/Write Functionality	Indicates read-only or read/write access in the Access File. Read/Write (W), the default, allows FOCUS MODIFY to update the RDBMS table. Read (R) is for reporting only. **Note:** If the Interface was installed for read-only access, this field displays an R and cannot be changed.
Date Display Format	Is a valid FOCUS USAGE date format for RDBMS columns described as dates. The default is YYMD.
Display Decimal when SCALE=0?	Yes (Y), the default, displays the decimal point for DECIMAL values with no fractional component (for example 123.); No (N) excludes the decimal point (for example, 123).
Use LABEL as Column Heading?	Selects FOCUS report column headings. No (N), the default, uses the column name for headings; Yes (Y) uses the RDBMS LABEL.
Use Remarks for FOCDEF?	Indicates whether to include RDBMS REMARKS as HELP for TableTalk. No (N), the default, excludes the remarks; Yes (Y) includes them. In MVS, you must specify a FOCDEF Target PDS in order to select Y.
Use Creator Name in AFD?	Indicates whether to include the creator name in the TABLENAME keyword of the Access File. Yes (Y), the default, includes the creator name; No (N) includes only the table name.

Using PFkeys From the Main Menu

You can implement the following functions from the Main Menu with PFkeys:

Key	**Function**	**Description**
PF1	Help	Accesses on-line help.
PF2	Refresh the List of Tables	Clears the existing list of tables maintained by AUTODB2/SQL for this session. When you select tables from the Main Menu, information is gathered from the system catalogs. Each time you change the selection criteria, the new tables are appended to the existing list. Pressing PF2 purges this list without exiting AUTODB2/SQL. You receive the message “Enter new table selection criteria” when the list is successfully purged. No message is displayed if you have not yet created a list of tables.
PF3	Exit	Ends the AUTODB2/SQL session and returns to FOCUS.

Key	**Function**	**Description**
PF4	Log Default Menu Parameters	Saves the values that you want to see displayed as defaults on the Main Menu in future executions of AUTODB2/SQL. In MVS, your defaults are saved in member DB2$PARM in the parm data set indicated on the menu. This data set is either: the PDS pre-allocated to DDNAME DB2$PARM, 'userid.FOCSQL.DATA', or the first data set allocated to DDNAME FOCSQL, whichever the system finds first (see *The MVS Parameter Log File* on page 6-14 for more detailed information on the search order). In CMS, your defaults are saved to filename SQL$PARM FOCSQL A. The following information is logged: • Creator Name • Table Name • Database Name (MVS only) • MFD Partitioned Data set Name (MVS only) • AFD Partitioned Data set Name (MVS only) • FOCDEF Partitioned Data set Name (MVS only) • Replace existing description • Read/Write Functionality • Date Display Format • Display Decimal when SCALE=0 • Use LABEL as Column Heading • Use REMARKS for FOCDEF • Use Creator Name in AFD **Note:** The values you enter must pass all validation tests in order for the new default values to be logged.
PF5 PF6	TED MFD TED AFD	Allows you to edit, using TED, the Master (PF5) or Access (PF6) File whose name you entered (as *mastername*) in the Master Filename entry field on the Main Menu. In MVS, you access member *mastername* in the data set entered as either Master Target PDS or Access Target PDS. In CMS, you access '*mastername* MASTER A' or '*mastername* FOCSQL A'. **Note:** You can edit these files even if they were not created with AUTODB2/SQL.

Key	Function	Description
PF9	Picture of MFD	Generates a diagram of the file entered in the Master Filename entry field on the Main Menu. After the picture is displayed, type any character and press Enter to return to the menu. To generate the picture in MVS, the *mastername* entered as Master Filename must be a member of a data set allocated to DDNAME MASTER (the Master Target PDS is not used).
PF10	Table List	Displays a list of all tables that meet the screening criteria provided in Creator Name, Table Name, and Database Name.

Note: The PFkey options are available only if the values you enter on the screen pass all validation tests. See *Common Errors* on page 6-18 for a discussion of common errors. See *Using PFkeys From non-Main Menu Screens* on page 6-12 for PFkey options available on other screens.

After you specify the appropriate values on the initial screen, press Enter. AUTODB2/SQL informs you that it is creating a list of tables with the following message:

```
     **==============================================================**
     **    AUTODB2 is retrieving TABLE  information from catalog.    **
     **    Please wait...                                            **
     **==============================================================**
```

It displays this message only for the first retrieval per AUTODB2/SQL session, or when the selection criteria have changed since the previous retrieval within the session. AUTODB2/SQL does not access the catalog a second time for the same immediate selections in a single session.

The Table Selection Screen

When table information retrieval is complete, choose the tables to include in your Master and Access Files from the Table Selection Screen:

```
Master:       Master File Generation Facility for DB2
AUTOEMP                  ==Table Selection==

Place an 'R' next to the Table to be the root of the Master.
Place  a 'C' next to all other Tables to be described as children.
Enter 'Y' next to all selected Tables that will be updated.

    Creator Name       DB2 Table Name    Select (R/C)  Write (Y/N)
    ------------      --------------   ----------- -----------
     USER1             ADDRESS                c              Y
     USER1             COURSE                                Y
     USER1             DEDUCT                                Y
     USER1             EMPINFO                r              Y
     USER1             FUNDTRAN                              Y
     USER1             PAYINFO                c              Y
     USER1             SALINFO                               Y






PF1=Help               PF3=End  PF4=Add Tables             PF7=Up  F8=Down
```

The Table Selection Screen displays, in alphabetical order, the names of all creator/table combinations that pass the selection criteria you provided on the Main Menu. You can choose up to 64 tables to include in the description. Identify the root table by placing r in its Select column. Choose all other tables to be included in the Master File by placing c in their Select columns. Indicate those tables that can be updated in this view by placing y in their Write columns.

Note: If the Interface was installed for read-only access, the Write column displays an N and cannot be changed.

The following message displays after you complete the table selections:

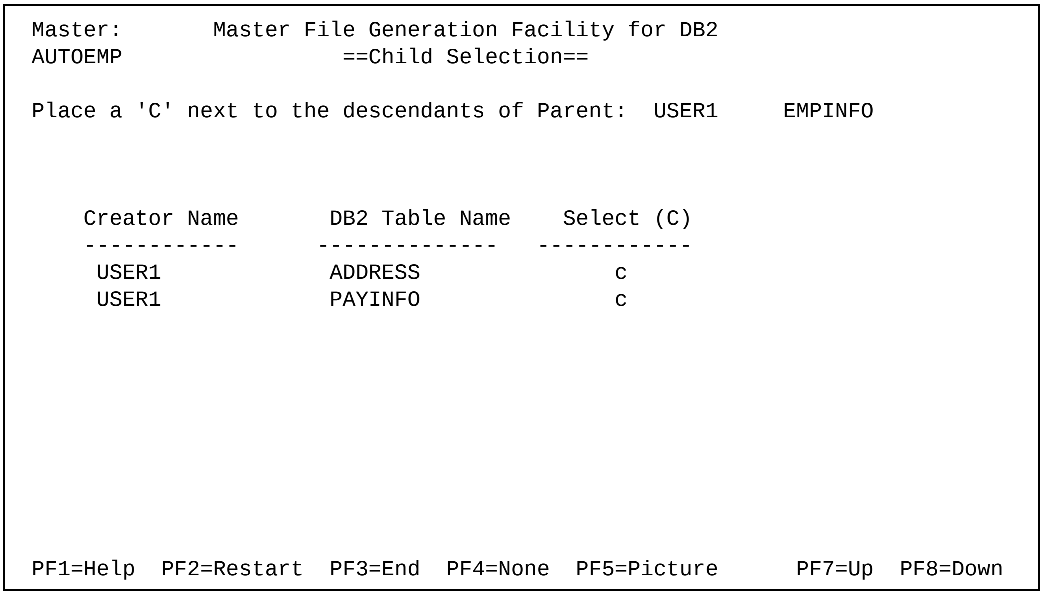
```
    **==============================================================**
    **    AUTODB2 is retrieving COLUMN information from catalog.   **
    **    Please wait...                                            **
    **==============================================================**
```

Note: If you selected only a root table, the preceding message does not appear; instead, AUTODB2/SQL creates the description and returns to the Main Menu.

AUTODB2/SQL retrieves column information from the catalog only the first time you select a particular table from the Table Selection Screen. You will not see this message when you make subsequent selections using the same tables. To erase this list and create a new one, return to the Main Menu with PF3, and restart with PF2.

The Child Selection Screen

When column information retrieval is complete, specify descendants, if any, for each table. To give you the opportunity to identify all parent-child relationships, every table on your list, in turn, presents its own Child Selection Screen:



```
Master:       Master File Generation Facility for DB2
AUTOEMP                 ==Child Selection==

Place a 'C' next to the descendants of Parent:  USER1     EMPINFO



    Creator Name       DB2 Table Name    Select (C)
    ------------      --------------   ------------
     USER1             ADDRESS               c
     USER1             PAYINFO               c









PF1=Help  PF2=Restart  PF3=End  PF4=None  PF5=Picture      PF7=Up  PF8=Down
```

If the parent segment named at the top of the screen has no children, press PF4. Otherwise, place c in the Select column for each table that is a child of the parent table named at the top of the screen, and press Enter. Next, AUTODB2 or AUTOSQL displays a Common Column Selection Screen for each parent-child pair so you can identify the columns they share.

The Common Column Selection Screen

For each parent-child pair, identify the primary keys from the parent table and the foreign keys from the related table on the Common Column Selection Screen:

```
Master:       Master File Generation Facility for DB2
AUTOEMP             ==Common Column Selection==

Number the primary key columns      | Number the corresponding foreign
(in sequence) in the Parent Table:  | key columns in the Child Table:
 USER1     EMPINFO                  | USER1     ADDRESS
                                    |
 Key    Column Name         Format  | Key     Column Name         Format
--------------------------------------------------------------------------
 1      EID                 A9      |  1      EID                 A9
        LN                  A15     |         AT                  A4
        FN                  A10     |         LN1                 A20
        HDT                 YYMD    |         LN2                 A20
        DPT                 A10     |         LN3                 A20
        CSAL                P9.2    |         ANO                 I9
        CJC                 A3      |
        OJT                 F9.2    |
        BONUS_PLAN          I9      |
                                    |
                                    |
                                    |

PF1=Help  PF2=Restart  PF3=End  PF4=Skip                   PF7=Up  PF8=Down
```

Number the key fields from each table in sequence (from 1 to 16). Each primary key field must have the same format as its corresponding foreign key field. Use PF4 to skip this table if it was selected in error. This does not affect previous or subsequent selections; however, the “skipped” segments appear in subsequent lists when you return to the Child Selection Screen to define any additional paths in the hierarchy.

AUTODB2/SQL guides you in describing the table relationships in top down, left to right order. Initially, it displays the root table and all of its selected children.

Completing the Description

After you complete the Common Column Selection Screen for the root table, AUTODB2/SQL displays a Child Selection Screen for the first child of the root. Assign children, if any, to this table.

Child selection continues down this first path until you press PF4 to select no descendants, or you exhaust the list of descendants. Only then does AUTODB2/SQL display a Child Selection Screen for the second child of the root. This process continues until all tables have been assigned children or all possible descendants have been exhausted.

At this point, AUTODB2 or AUTOSQL generates the Master and Access Files and returns to the Main Menu; the message “DESCRIPTION CREATED” displays at the bottom of this Status Screen with an indication, if necessary, of how many duplicate fieldnames were generated and how many unsupported datatypes were found.

Note: If duplicate fieldnames were created, either edit them to be unique, or qualify them with the segment name when referencing them in requests.

For information about the newly generated file descriptions, see *The Generated Descriptions* on page 6-16.

Using PFkeys From non-Main Menu Screens

For each screen other than the Main Menu and Table Selection screens, you can erase your current selections and back up one screen with PF2. Pressing PF3 returns you to the Main Menu with all selections intact. To erase your selections from the Table Selection Screen, return to the Main Menu with PF3, and then restart with PF2.

From the Child Selection Screen, use PF5 to generate a diagram of your file structure. The description is created on disk and remains there even if you end the program with PF3. After the picture is displayed, type any character and press the Enter key to return to the menu. To generate the picture in MVS, you must allocate the target master PDS to DDNAME MASTER. If a member with the selected master name exists in a data set concatenated in front of the target data set, the picture is generated from that member.

Retaining the List of Master Files Generated

AUTODB2/SQL maintains a temporary list of the Master Files generated during any one session. This list is refreshed at the beginning of each session and erased at the end of the session. You may retain the list by executing AUTODB2/SQL (either on-line or in the background) with the following syntax:

```
EX {AUTODB2|AUTOSQL} MFDLIST=Y
```

In MVS, the list resides in a temporary data set allocated to DDNAME AUTODB2L, with a disposition of MOD and record length of 114. In CMS, the list resides in the file AUTOSQLL FOCTEMP A with a record length of 26. It is the user's responsibility to free or erase this file when it is no longer required. The file layout is:

Length	**Columns**	**Description**
8	1 - 8	Master Filename
8	9 - 16	Number of duplicate fieldnames
5	17 - 21	Number of unsupported datatypes
5	22 - 26	Number of long decimal fields truncated. This entry, required in prior releases, exists for upward compatibility.
44	27 - 70	Master Target PDS name (MVS only)
44	71 - 114	Access Target PDS name (MVS only)

Changing the AUTODB2 Default Data Sets

With both AUTODB2 and AUTOSQL, you can save custom Main Menu defaults in a parameter log file for repeated use. For AUTODB2, you can also change the default data sets that display on the Main Menu.

To change the default MVS data set names that appear the first time you execute AUTODB2, customize the following lines of code near the top of the AUTODB2 FOCEXEC. These are the only permanent data sets that AUTODB2 uses. All other data sets are temporary and are named by the system.

```
-SET &DSNP0=&USERID ||'.FOCSQL.DATA                     ';
-SET &DSNM0=&USERID ||'.MASTER.DATA                     ';
-SET &DSNF0=&USERID ||'.FOCSQL.DATA                     ';
-SET &DSND0=&USERID ||'.FOCDEF.DATA                     ';
```

Note: You must preserve the length of the data set name for each variable (the result of concatenating the userid with the string between the single quotation marks) at 44 characters. If necessary, pad the name with blanks to maintain the correct length. You may not change any other lines in the FOCEXEC.

The MVS Parameter Log File

The parameter log file, if there is one, is always a data set allocated to DDNAME DB2$PARM in MVS. MVS identifies the parameter log file with the following steps:

1. If DDNAME DB2$PARM is allocated to a PDS prior to execution of AUTODB2, AUTODB2 uses member DB2$PARM as the parameter log file; if the member does not exist, AUTODB2 creates it. This allows the user to identify the “profile” data set prior to execution of AUTODB2 and is recommended for sites that have non-standard data set naming conventions.

   If this DDNAME is allocated to a sequential data set, AUTODB2 frees it, generates a message, and disables parameter logging.

   If DDNAME DB2$PARM is not allocated, AUTODB2 continues searching and attempts to allocate it in the steps that follow. AUTODB2 does not free this DDNAME upon exiting, assuming that it may be used again.

2. AUTODB2 allocates data set name 'userid.FOCSQL.DATA', the default name provided in the code as &DSNP0, as the parameter log file. If the default has been changed, it uses the new data set name. This assumes standard Information Builders naming conventions.

   This data set must be a PDS. If it is not, AUTODB2 displays a message and disables parameter logging.

3. If the first two steps fail to identify a parameter log file, AUTODB2 allocates the first data set allocated to DDNAME FOCSQL as the parameter log file. This assumes that the user’s data set is allocated first in the concatenation of data sets to DDNAME FOCSQL.

   If DDNAME FOCSQL is not allocated, AUTODB2 cannot function; it requires members DB2CAT and DB2SYSTB to be allocated to DDNAME FOCSQL. It displays a message and exits.

   If the data set allocated to DDNAME FOCSQL is sequential, AUTODB2 displays a message and disables parameter logging. **Note:** AUTODB2 cannot function properly if the data set is not a PDS.

Parameter logging assumes that the user has write access to the target data set. An attempt to log parameters to a data set without write access results in a security abend.

AUTODB2 and AUTOSQL in Batch Mode

You can create a single-table Master File by executing AUTODB2/SQL in the background with an argument list that supplies the values normally entered on the Main Menu. You can invoke AUTODB2 batch mode processing in the MVS batch, MSO, or TSO environments. The syntax is

```
EX AUTODB2 BATCH=Y,USERID=userid,MASTER=master,CREATOR=creator,
    TABLENAME=table [,option1=value1,...,optionk=valuek]
```

where:

userid

Is the default high-level qualifier for output data sets (MVS only).

master

Is the 1- to 8-character name of the resulting master file member.

creator

Is the 1- to 8-character name of the table's creator.

table

Is the 1- to 18-character name of the RDBMS table.

option1...optionk

Are options listed on the following chart

value1...valuek

Are acceptable values for the options.

Note: For more information, see the description of Main Menu entry fields in *The Main Menu* on page 6-4.

All options from the Main Menu are available with batch execution of AUTODB2 and AUTOSQL. Specify options on the command line by entering name=value pairs separated by commas. The list can extend over several lines. The following chart presents the available options:

Option Name	**Values**	**Description**	**Default**
REPLACE	Y=Yes,N=No	Replace existing description	N
FUNC	R=Read, W=Write	Read/Write Functionality	W
DISPDATE	date format	Date Display Format	YYMD
DECIMAL	Y=Yes,N=No	Display Decimal when SCALE=0?	Y
LABELS	Y=Yes,N=No	Use Labels as Column Heading?	N
REMARKS	Y=Yes,N=No	Use Remarks for FOCDEF?	N
CREATAFD	Y=Yes,N=No	Use Creator Name in AFD?	Y
MFDLIST	Y=Yes,N=No	Store list of MFDs created per session	N
MASTERDATA	dsn	Master Target PDS (MVS only)	&USERID.MASTER.DATA
FOCSQLDATA	dsn	Access Target PDS (MVS only)	&USERID.FOCSQL.DATA
FOCDEFDATA	dsn	FOCDEF Target PDS (MVS only)	&USERID.FOCDEF.DATA

The Generated Descriptions

The AUTODB2 and AUTOSQL facilities generate Master and Access Files containing the declarations described in Chapter 4, *Describing Tables to FOCUS*, and Chapter 5, *Multi-Table Structures*. This section discusses those cases in which AUTODB2 and AUTOSQL supply a different keyword or value. Minor changes may be required in some situations. As a rule, you can use the generated file descriptions, without changes, immediately after the AUTODB2 or AUTOSQL session.

In the Master File:

- The FILENAME value is the member name or file name you specified on the initial screen.
- The SEGNAME value is the RDBMS table name. The SEGTYPE value is always S0 for the root segment. For descendant segments it is either S0 or U, depending on the relationships between the various key fields.
- Field declarations for primary key columns are listed first, followed by those for non-key fields. FIELDNAME and ALIAS values are the full RDBMS column names.

  AUTODB2 and AUTOSQL support the TIME, TIMESTAMP, GRAPHIC and VARGRAPHIC datatypes. TIME and TIMESTAMP datatypes are treated as alphanumeric strings in the FOCUS Master File. GRAPHIC and VARGRAPHIC datatypes are described with an ACTUAL attribute of Kn, and with a USAGE attribute of A(2n+2).

  LONG VARGRAPHIC datatypes are not supported.

  AUTOSQL also supports DBAINT and DBAHW, two internal SQL/DS datatypes used in some system catalog tables. They are described with an ACTUAL attribute of I4 and should be used for read-only access.
- Field lengths for USAGE and ACTUAL formats are calculated automatically. The generated field declarations omit the keywords USAGE and ACTUAL; only the values appear. The ACTUAL field length for DECIMAL(p,s) columns with missing data is P8 if $p \leq 15$, P((p+1)/2) if $p > 15$. Columns with RDBMS DATE datatypes are described with an ACTUAL attribute of DATE. Columns with VARCHAR datatypes longer than 254 characters are described with an ACTUAL attribute of text (TX).
- The MISSING parameter is included and is set ON if the RDBMS table definition allows NULLs.

In the Access File:

- The SEGNAME value matches the corresponding SEGNAME value in the Master File.
- If Y was entered on the Main Menu for the field “Use Creator Name in AFD?”, the creator name and the table name compose the TABLENAME value.
- KEYS and WRITE values are supplied.

  AUTODB2 and AUTOSQL use existing RDBMS indexes to determine the KEYS value and the order in which to place fields in the Master File. KEYS is set to zero if a unique index does not exist or if an RDBMS view is described.

  Note: In the case of a view, the unique indexes are described on the underlying tables, not on the view; they may not be valid as a primary key for the view. If possible, edit the Master and Access Files to reflect the index structure of the view's base tables.
- The optional keyword DBSPACE is omitted.

You may need to edit the generated Master File or its corresponding Access File if:

- You prefer fieldnames different from the generated names.
- There are SQL datatypes in the tables or views that the Interface does not support. A warning about unsupported datatypes appears on the Status Screen.

  Field declarations are not generated for columns with unsupported datatypes. You can add the field declarations yourself; consult *ACTUAL Format Conversion Chart* in Chapter 4, *Describing Tables to FOCUS*, for more information on describing them.

  To identify the unsupported columns, check the system catalog, SYSCOLUMNS, with a FOCUS report request. (See *Querying the RDBMS SYSCOLUMNS Catalog* in Appendix A, *Additional Topics*, for an example.)

  AUTODB2 and AUTOSQL include all datatypes that FOCUS can access.
- A unique index does not exist, or you want to use a different index from the selected one. If the table or view has a primary key, specify a value greater than zero for the KEYS keyword and rearrange the fields in the Master File so that the columns comprising the primary key are described first.
- Certain fields are classified for security reasons. To add security, delete field declarations or include the FOCUS DBA attribute.
- You require options such as DEFINE statements and KEYORDER.

Common Errors

AUTODB2 and AUTOSQL provide an easy, straightforward approach for defining tables and views to FOCUS. Most errors occur at the initial screen level and result in error messages. Many fields trigger validation tests that prompt you for a valid entry. Common errors include:

- Leaving the Master Filename entry field blank on the initial screen.
- Specifying an invalid userid for the Creator Name entry field.
- Leaving the Table Name entry field blank or specifying a nonexistent table or view. In these cases, no records are retrieved and you return to the Main Menu.
- Specifying an invalid character for the Read/Write Functionality entry field.
- Specifying a duplicate member name or file name for the Master Filename entry field. If the REPLACE option is specified, AUTODB2 and AUTOSQL overwrite the existing members or files.

At the Status Screen level, you may see one of the following warning messages:

- n DUPLICATE. The tables or views contain duplicate column names. You can edit these field names to make them unique, or qualify them with the segment name when referencing them in requests.

- n UNSUPPORTED. The table or view contains some columns with datatypes not supported by FOCUS (LONG VARGRAPHIC, for example).

Other circumstances can also affect processing:

- For AUTODB2, permanent PDSs are too small to receive the generated file descriptions as members.
- For AUTOSQL, there is not enough disk space on the A disk.
- You do not have proper authorization for SELECT privileges on the system catalog tables.

AUTODB2 Sample Session

This sample session executes the AUTODB2 facility to generate Master and Access Files based on the existing EMPINFO, ADDRESS, and PAYINFO tables; it assigns the name AUTOEMP to the new file descriptions.

Note: Lowercase values represent user input.

First, execute the AUTODB2 FOCEXEC from the FOCUS command line:

```
>ex autodb2
```

AUTODB2 displays the Main Menu with its pre-set defaults. Type autoemp in the Master Filename entry field and press Enter:

```
Main Menu     Master File Generation Facility for DB2
      Master Filename ================> autoemp

      Information extracted from:
        Creator Name (or *) ==========> user1
        Table Name   (or *) ==========> *
        Database Name ================> *
      Description will be a member of:
        Master Target PDS => USER1.MASTER.DATA
        Access Target PDS => USER1.FOCSQL.DATA
        FOCDEF Target PDS => USER1.FOCDEF.DATA

        Replace Existing Description?=> Y       (Y/N)
        Read/Write Functionality =====> W       (R=Read,W=Write)
        Date Display Format ==========> YYMD
        Display Decimal when SCALE=0?=> Y       (Y/N)
        Use LABEL as Column Heading? => N       (Y/N)
        Use Remarks for FOCDEF? ======> N       (Y/N)
        Use Creator Name in AFD? =====> Y       (Y/N)
        Use Long Fieldnames? =========> Y       (Y/N)
        Parm File => USER1.FOCSQL.DATA

PF1=Help PF2=Restart PF3=Exit PF4=Log PF5=MFD PF6=AFD PF9=Picture PF10=List
```

The default values in the creator, table, and database name fields establish a pattern that selects all tables for creator USER1. AUTODB2 indicates that it is retrieving table information from the catalog:

```
**=============================================================**
**    AUTODB2 is retrieving TABLE  information from catalog.   **
**    Please wait...                                           **
**=============================================================**
```

Next, AUTODB2 presents the Table Selection Screen. Designate EMPINFO as the root by placing r in its Select column; designate ADDRESS and PAYINFO as children by placing c in their Select columns:

```
Master:       Master File Generation Facility for DB2
AUTOEMP                  ==Table Selection==

Place an 'R' next to the Table to be the root of the Master.
Place  a 'C' next to all other Tables to be described as children.
Enter 'Y' next to all selected Tables that will be updated.

    Creator Name       DB2 Table Name    Select (R/C)  Write (Y/N)
    ------------       --------------    ------------  -----------
     USER1             ADDRESS               c              Y
     USER1             COURSE                               Y
     USER1             DEDUCT                               Y
     USER1             EMPINFO               r              Y
     USER1             FUNDTRAN                             Y
     USER1             PAYINFO               c              Y
     USER1             SALINFO                              Y






PF1=Help               PF3=End  PF4=Add Tables              PF7=Up  F8=Down
```

AUTODB2 indicates that it is gathering column information about those tables:

```
    **==============================================================**
    **    AUTODB2 is retrieving COLUMN information from catalog.    **
    **    Please wait...                                            **
    **==============================================================**
```

Now, AUTODB2 displays a Child Selection Screen for the root table, EMPINFO. To define ADDRESS and PAYINFO as children of EMPINFO, place c in their Select columns:

```
Master:       Master File Generation Facility for DB2
AUTOEMP                 ==Child Selection==

Place a 'C' next to the descendants of Parent:  USER1     EMPINFO



    Creator Name       DB2 Table Name    Select (C)
    ------------       --------------    ------------
     USER1             ADDRESS               c
     USER1             PAYINFO               c











PF1=Help  PF2=Restart  PF3=End  PF4=None  PF5=Picture      PF7=Up  PF8=Down
```

In a more complicated structure, some of the child tables could be descendants of other child tables; for example, PAYINFO could be a child of EMPINFO, and ADDRESS could be a child of PAYINFO. When necessary, AUTODB2 displays additional Child Selection Screens. However, in this example, all tables in the structure have been accounted for, so no additional Child Selection Screens display.

Next, identify the primary key from EMPINFO and the foreign key from ADDRESS on the Common Column Selection Screen. Identify EID as the primary key for the EMPINFO table by placing 1 in its Key column. Indicate that EID is the corresponding foreign key in the ADDRESS table by placing 1 in its Key column:

```
Master:       Master File Generation Facility for DB2
AUTOEMP              ==Common Column Selection==

Number the primary key columns      | Number the corresponding foreign
(in sequence) in the Parent Table:  | key columns in the Child Table:
 USER1     EMPINFO                  | USER1     ADDRESS
                                    |
 Key    Column Name         Format  | Key     Column Name         Format
-----------------------------------------------------------------------
 1      EID                 A9      |  1      EID                 A9
        LN                  A15     |         AT                  A4
        FN                  A10     |         LN1                 A20
        HDT                 YYMD    |         LN2                 A20
        DPT                 A10     |         LN3                 A20
        CSAL                P9.2    |         ANO                 I9
        CJC                 A3      |
        OJT                 F9.2    |
        BONUS_PLAN          I9      |
                                    |
                                    |
                                    |

PF1=Help  PF2=Restart  PF3=End  PF4=Skip                  PF7=Up  PF8=Down
```

Since no other fields participate in the relationship, press Enter.

AUTODB2 displays another Common Column Selection Screen, for the EMPINFO and PAYINFO tables. Identify the primary and foreign keys that relate those two tables:

```
Master:       Master File Generation Facility for DB2
AUTOEMP             ==Common Column Selection==

Number the primary key columns      | Number the corresponding foreign
(in sequence) in the Parent Table:  | key columns in the Child Table:
 USER1     EMPINFO                  | USER1     PAYINFO
                                    |
 Key    Column Name         Format  | Key     Column Name         Format
--------------------------------------------------------------------------
 1      EID                 A9      |  1      EID                 A9
        LN                  A15     |         DI                  YYMD
        FN                  A10     |         PI                  F9.2
        HDT                 YYMD    |         SAL                 D12.2
        DPT                 A10     |         JBC                 A3
        CSAL                P9.2    |
        CJC                 A3      |
        OJT                 F9.2    |
        BONUS_PLAN          I9      |
                                    |
                                    |
                                    |

PF1=Help  PF2=Restart  PF3=End  PF4=Skip                  PF7=Up  PF8=Down
```

Press the Enter key.

At this point, AUTODB2 creates the Master and Access Files. The Main Menu includes the “DESCRIPTION CREATED” message at the bottom of the screen:

```
Main Menu     Master File Generation Facility for DB2
      Master Filename ================> AUTOEMP

      Information extracted from:
        Creator Name (or *) ==========> USER1
        Table Name   (or *) ==========> *
        Database Name ================> *
      Description will be a member of:
        Master Target PDS => USER1.MASTER.DATA
        Access Target PDS => USER1.FOCSQL.DATA
        FOCDEF Target PDS => USER1.FOCDEF.DATA

        Replace Existing Description?=> Y       (Y/N)
        Read/Write Functionality =====> W       (R=Read,W=Write)
        Date Display Format ==========> YYMD
        Display Decimal when SCALE=0?=> Y       (Y/N)
        Use LABEL as Column Heading? => N       (Y/N)
        Use Remarks for FOCDEF? ======> N       (Y/N)
        Use Creator Name in AFD? =====> Y       (Y/N)
        Use Long Fieldnames? =========> Y       (Y/N)
        Parm File => USER1.FOCSQL.DATA
 DESCRIPTION CREATED - 3 DUPLICATE
PF1=Help PF2=Restart PF3=Exit PF4=Log PF5=MFD PF6=AFD PF9=Picture PF10=List
```

Three duplicate field names were created. Either edit them to be distinct, or qualify them with the segment name in requests.

To see a picture of the structure created, press PF9:

```
 NUMBER OF ERRORS=     0
 NUMBER OF SEGMENTS=   3  ( REAL=    3  VIRTUAL=   0 )
 NUMBER OF FIELDS=    20  INDEXES=   0  FILES=     1
 TOTAL LENGTH OF ALL FIELDS=  168
SECTION 01
             STRUCTURE OF SQLDS    FILE AUTOEMP  ON 05/25/95 AT 11.01.09

         EMPINFO
 01      S0
**************
*EID         **
*LN          **
*FN          **
*HDT         **
*            **
***************
 **************
       I
       +-----------------+
       I                 I
       I ADDRESS         I PAYINFO
 02    I S0        03    I S0
**************    **************
*EID         **   *EID         **
*AT          **   *DI          **
*LN1         **   *PI          **
*LN2         **   *SAL         **
*            **   *            **
***************   ***************
 **************    **************

TYPE ANY CHARACTER AND PRESS ENTER TO CONTINUE  >
```

As indicated on the screen, type any character and press Enter to continue.

To edit The AUTOEMP Master File (member AUTOEMP in data set USER1.MASTER.DATA), press PF5:

```
$$$ CREATED BY AUTODB2 ON 05/25/95 AT 10.54.56 BY USER1
FILENAME=AUTOEMP,SUFFIX=SQLDS,$

SEGNAME=EMPINFO,SEGTYPE=S0,$
FIELD=EID              ,EID               ,A9     ,A9   ,MISSING=OFF,$
FIELD=LN               ,LN                ,A15    ,A15  ,MISSING=OFF,$
FIELD=FN               ,FN                ,A10    ,A10  ,MISSING=OFF,$
FIELD=HDT              ,HDT               ,YYMD   ,DATE ,MISSING=OFF,$
FIELD=DPT              ,DPT               ,A10    ,A10  ,MISSING=ON,$
FIELD=CSAL             ,CSAL              ,P9.2   ,P4   ,MISSING=OFF,$
FIELD=CJC              ,CJC               ,A3     ,A3   ,MISSING=OFF,$
FIELD=OJT              ,OJT               ,F9.2   ,F4   ,MISSING=ON,$
FIELD=BONUS_PLAN       ,BONUS_PLAN        ,I9     ,I4   ,MISSING=OFF,$

SEGNAME=ADDRESS,SEGTYPE=S0,PARENT=EMPINFO,$
FIELD=EID              ,EID               ,A9     ,A9   ,MISSING=OFF,$
FIELD=AT               ,AT                ,A4     ,A4   ,MISSING=OFF,$
FIELD=LN1              ,LN1               ,A20    ,A20  ,MISSING=OFF,$
FIELD=LN2              ,LN2               ,A20    ,A20  ,MISSING=OFF,$
FIELD=LN3              ,LN3               ,A20    ,A20  ,MISSING=OFF,$
FIELD=ANO              ,ANO               ,I9     ,I4   ,MISSING=OFF,$

SEGNAME=PAYINFO,SEGTYPE=S0,PARENT=EMPINFO,$
FIELD=EID              ,EID               ,A9     ,A9   ,MISSING=OFF,$
FIELD=DI               ,DI                ,YYMD   ,DATE ,MISSING=OFF,$
FIELD=PI               ,PI                ,F9.2   ,F4   ,MISSING=OFF,$
FIELD=SAL              ,SAL               ,D12.2  ,D8   ,MISSING=OFF,$
FIELD=JBC              ,JBC               ,A3     ,A3   ,MISSING=OFF,$
$DUPLICATE=EID              ,COUNT= 3,SEGNAME=EMPINFO
$DUPLICATE=EID              ,COUNT= 3,SEGNAME=ADDRESS
$DUPLICATE=EID              ,COUNT= 3,SEGNAME=PAYINFO
```

To exit from the FOCUS TED editor, press PF3, or type FILE to save any changes you made.

To edit the AUTOEMP Access File (member AUTOEMP in data set USER1.FOCSQL.DATA), press PF6. Notice the KEYFLD and IXFLD values that define the embedded JOIN:

```
$$$ CREATED BY AUTODB2 ON 05/25/95 AT 10.54.56 BY USER1
$$$ FILENAME=AUTOEMP,SUFFIX=SQLDS,$

SEGNAME=EMPINFO,TABLENAME='"USER1"."EMPINFO"',
KEYS=01,WRITE=YES,KEYORDER=LOW,$

SEGNAME=ADDRESS,TABLENAME='"USER1"."ADDRESS"',
KEYS=02,WRITE=YES,KEYORDER=LOW,
KEYFLD=EID,
IXFLD=EID,$

SEGNAME=PAYINFO,TABLENAME='"USER1"."PAYINFO"',
KEYS=02,WRITE=YES,KEYORDER=LOW,
KEYFLD=EID,
IXFLD=EID,$
```

To exit from the FOCUS TED editor, press PF3, or type FILE to save any changes you made.

The CREATE FILE Command

The FOCUS CREATE FILE command uses existing FOCUS Master and Access Files to generate new RDBMS tables and, possibly, unique indexes.

Before issuing the FOCUS CREATE FILE command, make sure you have:

- RDBMS GRANT authority to create tables (as described in Chapter 3, *Security*).
- A Master File. Field declarations describing the primary key column(s) must be listed first.
- An Access File. If KEYS is greater than zero, a unique index will be created. To create the index in descending order, set KEYORDER to HIGH.

  If the Access File does not include a DBSPACE value, you can issue the SET DBSPACE command to establish a default tablespace or dbspace for the duration of the FOCUS session. (Consult Chapter 4, *Describing Tables to FOCUS*, for the DBSPACE attribute and Chapter 11, *Environmental Commands*, for the SET DBSPACE command.)

  If you do not issue the SET DBSPACE command, CREATE FILE uses the Interface installation default. If your site did not establish a default during installation:

  - For DB2, the table is placed in the default DB2 database, DSNDB04, and DB2 dynamically creates the tablespace.
  - For SQL/DS, the table is placed in a private dbspace owned by the user. If none exists, an error message is returned.

From the FOCUS command level, enter

```
CREATE FILE name
```

where:

`name`
Is the name of the Master and Access Files.

When the table is successfully generated, the FOCUS command level prompt (>) appears.

The Interface generates one table and one unique index (provided the KEYS parameter is not 0) for every segment declaration in a multi-table Master File. It accomplishes this in a single logical unit of work, so if one of the tables already exists, it does not create the others. That is, the FOCUS CREATE FILE command does not overwrite an existing RDBMS table as it may do for a FOCUS database.

Note: You can control index space parameters with the Interface SET IXSPACE command described in Chapter 11, *Environmental Commands*.

You have two choices if an SQL error occurs:

- Change the value of the TABLENAME keyword in the Access File and reissue the CREATE FILE command.
- Discard the existing table with the SQL DROP command and reissue the CREATE FILE command.

You can also create tables by:

- Issuing the native SQL CREATE TABLE command from within the FOCUS environment. Consult Chapter 9, *Direct SQL Passthru*, for Direct SQL Passthru.
- Using the HOLD FORMAT SQL option in a report request. See *Creating Extract Files on the RDBMS* in Chapter 8, *Advanced Reporting Techniques*, for information about extract files.

CREATE FILE Example

The following DB2 session illustrates table creation. The member name for the pair of file descriptions is EMPINFO. In order to trace the process, the example allocates the FSTRACE facility. (For more information about the Interface trace facilities, see Appendix D, *Tracing Interface Processing*.)

Since the EMPINFO Master and Access Files exist (see *EMPINFO Sample* in Appendix C, *File Descriptions and Tables*), the CREATE FILE command can create the EMPINFO table:

```
>  > tso alloc f(fstrace) da(*) lrecl(80) recfm(f)
>  > create file empinfo
```

The trace displays the SQL statements issued by the Interface:

```
>  > create file empinfo
 SQL: ************************************
 SQL: *** COMMAND OPEN   count = 0 *******
 SQL: ***
 SQL: ************************************
 SQL: *** EXECUTE IMMEDIATE **************
 SQL: ***
 SQL:  CREATE TABLE "EMPINFO"( EID CHAR (009) NOT NULL ,LN CHAR (015) NOT
 SQL:  NULL ,FN CHAR (010) NOT NULL ,HDT DATE  NOT NULL ,DPT CHAR (010),C
 SQL: SAL DECIMAL(07, 02) NOT NULL ,CJC CHAR (003) NOT NULL ,OJT REAL ,BO
 SQL: NUS_PLAN INTEGER  NOT NULL ) IN PUBLIC.SPACE0
 SQL:            **** SQLCA Results ****
 SQL: Sqlcode:  0
 SQL: Sqlerrp:  DSN
 SQL: Sqlerrd:  d1=0 d2=0 d3=0 d4=-1 d5=0 d6=0
 SQL: ************************************
 SQL: *** EXECUTE IMMEDIATE **************
 SQL: ***
 SQL:  CREATE UNIQUE INDEX "EMPINFOIX" ON "EMPINFO" (EID  ASC)
 SQL:            **** SQLCA Results ****
 SQL: Sqlcode:  0
 SQL: Sqlerrp:  DSN
 SQL: Sqlerrd:  d1=0 d2=0 d3=0 d4=-1 d5=0 d6=0
 SQL: ************************************
 SQL: *** COMMIT WORK   ******************
 SQL: ***
 SQL:            **** SQLCA Results ****
 SQL: Sqlcode:  0
 SQL: Sqlerrp:  DSN
 SQL: ***
 SQL: *** COMMAND CLOSE  count = 0 *******
 SQL: ************************************
```

The Interface generates one SQL CREATE TABLE command that consists of:

- Column information from the field declarations in the Master File.
- Table information from the TABLENAME value in the Access File.

The new table resides in tablespace PUBLIC.SPACE0.

Since the KEYS value in the Access File is greater than zero, the Interface issues the SQL CREATE UNIQUE INDEX command. The first n fields in the Master File and the KEYS value provide the required information.

The index EMPINFOIX is created in ascending order, the RDBMS default. Its name is composed of the table name and the suffix IX. The parentheses around the EID field from the Master File indicate that it is the column to be indexed.

If no SQL errors result from table or index creation, the Interface issues the SQL COMMIT WORK command to permanently define the table and its index. If an error occurs, the Interface issues an SQL ROLLBACK WORK command. The ROLLBACK WORK command returns the RDBMS catalog tables to their original state, and table generation stops.

In this example there are no errors, since each generated SQL statement returns an SQLCODE of 0. The Interface issues the SQL COMMIT WORK command to permanently define the table and its index.

7 The Interface Optimizer

Topics:

Optimizing Requests

Optimization is the process in which the Interface translates the projection, selection, join, sort, and aggregation operations of a FOCUS report request into their SQL equivalents and passes them to the RDBMS for processing.

Interface optimization allows the RDBMS to perform the work for which it is best suited, reducing the volume of RDBMS-to-FOCUS communication and improving response time. It also enables the RDBMS to exploit its own internal optimization techniques.

Syntax How to Invoke Optimization

To invoke the optimization process, enter the following Interface command at the FOCUS command level

```
SQL [target_db] SET {OPTIMIZATION|SQLJOIN} setting
```

where:

`target_db`

Is the target RDBMS. Valid values are DB2 or SQL/DS. Omit if you previously issued the SET SQLENGINE command.

`SQLJOIN`

Is a synonym for OPTIMIZATION.

`setting`

Is the optimization setting. Valid values are as follows:

`OFF` instructs the Interface to create SQL statements for simple data retrieval from each table. FOCUS handles all aggregation, sorting, and joining in your address space or virtual machine to produce the report.

`ON` instructs the Interface to create SQL statements that take advantage of RDBMS join, sort, and aggregation capabilities. It is compatible with previous releases in regard to the multiplicative effect: misjoined unique segments and multiplied lines in PRINT and LIST based report requests do not disable optimization (see *RDBMS and FOCUS Join Management* on page 7-3). Other cases of the multiplicative effect invoke the Interface-managed native join logic described in *Optimizing Joins* on page 7-6. ON is the default.

`FOCUS` passes join logic to the RDBMS only when the results will be the same as from a FOCUS-managed request. Misjoined unique segments, the multiplicative effect, and multiplied lines in PRINT and LIST based report requests (see *RDBMS and FOCUS Join Management* on page 7-3) invoke the Interface-managed native join logic described in *Optimizing Joins* on page 7-6.

`SQL` passes join logic to the RDBMS in all possible cases. The multiplicative effect does not disable optimization, even in cases involving aggregation (SUM, COUNT). Does not pass join logic to the RDBMS for tables residing on multiple subsystems and for tables residing on multiple DBMS platforms.

You can use traces FSTRACE3 and FSTRACE4 to evaluate the SQL statements generated by the Interface. FSTRACE3 verifies whether the Interface passed aggregation and join operations to the RDBMS. FSTRACE4 displays the SQL SELECT statements that the Interface generates from report requests. A single SQL SELECT statement indicates RDBMS-managed join processing; two or more indicate FOCUS-managed join processing. For information about Interface trace facilities, see Appendix D, *Tracing Interface Processing*.

RDBMS and FOCUS Join Management

When the RDBMS manages a join, it first generates a Cartesian product of the tables, then applies any screening conditions to the resulting rows (including any join conditions), then applies the SELECT list, and, finally, calculates column function values and expressions.

In contrast, when FOCUS manages a join, it first reads segment instances from the top to the bottom of the file structure.

Because FOCUS views RDBMS tables as segments, in very rare situations the RDBMS could return more instances of a row than FOCUS would if it were processing similar data from a FOCUS database or a sequential file. This difference is called the multiplicative effect.

FOCUS implements a dynamic or embedded join between RDBMS tables (segments, to FOCUS) based on a set of pairs of “from/to” fields. The “to” fields are called the foreign key.

The KEYS attribute in the Access File defines the number of primary key fields (columns) in a segment; KEYS = n indicates that the first n fields in the segment comprise the primary key. If the foreign key fields for a child segment or cross-referenced file do not completely cover its primary key, multiple rows in the child segment (table) may correspond to a single row in the parent segment. Therefore, the RDBMS-generated Cartesian product may contain multiple instances of a single parent segment row; the parent segment is multiplied. By definition, all ancestors of the multiplied segment are also multiplied.

A unique segment is defined as having exactly one instance corresponding to an instance of its parent segment. If its foreign key fields do not cover its primary key, this is not necessarily the case, even though the join was specified to FOCUS as unique. A segment in this situation is called misjoined.

Even if a unique segment is misjoined, however, the Interface does not consider it as causing its parent's multiplication. The Interface processes a “unique segment misjoined” condition separately from the multiplicative effect; it issues a warning message, and, depending on the current optimization setting, may or may not disable optimization.

In most cases, the Interface can prevent the multiplicative effect, as described in *Optimizing Joins* on page 7-6 and *Optimizing Aggregation* on page 7-11.

Example How Optimization Affects SQL Requests Passed to the RDBMS

The example in this section demonstrates the differences between SQL statements generated with and without optimization; the report request joins tables EMPINFO and FUNDTRAN with traces FSTRACE3 and FSTRACE4 allocated.

When optimization is disabled, the Interface generates two SELECT statements. The first SELECT retrieves any rows from the EMPINFO table that have the value MIS in the DEPARTMENT column. For each EMPINFO row, the second SELECT retrieves rows from the cross-referenced FUNDTRAN table, resolving the parameter marker (?) with the value of the host field (EMP_ID). Both SELECT statements retrieve answer sets, but FOCUS performs the join, sort, and aggregation operations:

```
sql db2 set optimization off
 >  > join emp_id in empinfo to all who in fundtran as j1
 >  > table file empinfo
 > sum ave.current_salary ed_hrs by who by last_name
 > if department eq 'mis'
 > end
   (FOC2510) FOCUS-MANAGED JOIN SELECTED FOR FOLLOWING REASON(S):
   (FOC2511) DISABLED BY USER
   (FOC2590) AGGREGATION NOT DONE FOR THE FOLLOWING REASON:
   (FOC2592) RDBMS-MANAGED JOIN HAS BEEN DISABLED
     SELECT T1.EID,T1.LN,T1.DPT,T1.CSAL,T1.OJT FROM
    "USER1"."EMPINFO" T1 WHERE (T1.DPT = 'MIS') FOR FETCH ONLY;
     SELECT T2.EID FROM "USER1"."FUNDTRAN" T2 WHERE (T2.EID = ?)
    FOR FETCH ONLY;
```

With optimization enabled, the Interface generates one SELECT statement that incorporates the join, sort, and aggregation operations. The RDBMS manages and processes the request; FOCUS only formats the report.

```
sql db2 set optimization on
 >  > join emp_id in empinfo to all who in fundtran as j1
 >  > table file empinfo
 > sum ave.current_salary ed_hrs by who by last_name
 > if department eq 'mis'
 > end
 AGGREGATION DONE ...
     SELECT T2.EID,T1.LN, AVG(T1.CSAL), SUM(T1.OJT) FROM
    "USER1"."EMPINFO" T1,"USER1"."FUNDTRAN" T2 WHERE (T2.EID =
    T1.EID) AND (T1.DPT = 'MIS') GROUP BY T2.EID,T1.LN ORDER BY
    T2.EID,T1.LN FOR FETCH ONLY;
```

Both situations produce the same report.

Optimization Logic

Optimization means passing the following tasks to the RDBMS:

- Record selection and projection
- Joins
- Sorting
- Aggregation

Aside from record selection and projection, the Interface does not offload these functions to the RDBMS if you set OPTIMIZATION OFF. When OPTIMIZATION is not OFF, the Interface evaluates each report request for the existence of joins, sort operations, and finally aggregation operations. It automatically invokes optimization for record selection and projection operations.

Optimizing Record Selection and Projection

Regardless of the OPTIMIZATION setting, the Interface may pass record selection and projection to the RDBMS; it is always more efficient to do so.

Record Selection

The Interface can translate all forms of FOCUS record selection to predicates of an SQL WHERE clause, except those that contain:

- Certain FOCUS DEFINE fields (see *Optimizing DEFINE Fields* on page 7-11).
- EDIT to perform datatype conversions, EDIT of an alphanumeric field that was the result of such a conversion, or EDIT of a DEFINE field.
- LIKE with fields created using DEFINE or COMPUTE.
- DATE fields with formats other than YMD or YYMD.

Starting with Version 7.0, the Interface optimizes screening conditions based on DEFINE fields that derive their values from a single segment of the join structure. The Interface optimizes these screening conditions as long as you do not set OPTIMIZATION OFF, even if it has disabled join optimization in your request.

When using LIKE in a WHERE clause, make sure any constant in the LIKE predicate either:

- Has the same length as the field used in the comparison.
- Is padded with blanks or underscore characters (_) to maintain the appropriate length.
- Contains the % wildcard character to denote any sequence of characters.

Escape characters in the LIKE predicate are optimized. See the *FOCUS for IBM Mainframe User's Manual* for a discussion of LIKE.

If you fail to specify a wildcard pattern in the LIKE predicate, the WHERE clause passes an equality predicate to the RDBMS instead of the LIKE predicate. For example

```
WHERE EID LIKE 'A'
```

will generate

```
WHERE (T1.EID = 'A')
```

not:

```
WHERE (T1.EID LIKE 'A')
```

In addition, because of unpredictable comparisons between VARCHAR datatypes, the WHERE clause is not passed to the RDBMS if both of the following conditions are true:

- The specified pattern is not equal in length to the column used in the comparison.
- The pattern contains one or more wildcard characters but does not terminate with the percent character (%).

In this case, FOCUS performs the screening condition on all rows returned from the RDBMS.

Projection

In relational terminology, projection is the act of retrieving any subset of the available columns from a table. The Interface implements projection by retrieving only those columns referenced in a TABLE request. Projection reduces the volume of data returned from the RDBMS, which, in turn, improves application response time.

Optimizing Joins

This discussion applies to joins invoked either by the FOCUS dynamic JOIN command (see *The Dynamic JOIN Command* in Chapter 8, *Advanced Reporting Techniques*) or by the embedded join facility (see Chapter 5, *Multi-table Structures*).

When you invoke OPTIMIZATION, the Interface attempts to build a single SQL statement. In order for the Interface to create one SQL statement that incorporates an RDBMS join on primary and foreign keys:

- The tables must share a single retrieval path in the joined structure; multiple paths are not permitted.
- Except for the lowest level segment referenced in the request, primary keys must exist (KEYS > 0 in the Access File) for all segments referenced in a report request that includes:
    - Aggregation operators such as SUM or COUNT, when the OPTIMIZATION setting is ON or FOCUS.

- Multiple FST. or LST. operators on a segment that could return more than one record per sort break, regardless of the OPTIMIZATION setting.

  In prior releases, SUM or COUNT operations at any level other than the lowest level in the join structure caused the RDBMS to duplicate data. In effect, each host row was replicated for each associated row in the cross-referenced table; this duplication of data is known as the multiplicative effect. In prior releases, detection of the multiplicative effect disabled optimization unless the user set OPTIMIZATION to SQL (see *Optimizing Requests* on page 7-1).

  Starting with Version 7.0, when all segments other than the lowest level segment in the request have primary keys, the Interface passes such joins to the RDBMS with an SQL ORDER BY clause on all columns of each segment's primary key, in top to bottom order. The resulting sort on the returned answer set enables the Interface to eliminate duplicate rows before passing the data to FOCUS. This technique is called Interface-managed native join optimization.

  In some cases, Interface-managed joins may be less efficient than FOCUS-managed joins from prior releases. To invoke a FOCUS-managed join, set OPTIMIZATION to OFF.

  When the Interface manages join optimization, it does not optimize aggregation or sorting, as described in *Optimizing Sorts* on page 7-9 and *Optimizing Aggregation* on page 7-11. This behavior is consistent with that of prior releases when the join was not passed. Expressions based on a single segment in the structure, however, are passed (see *Optimizing Record Selection and Projection* on page 7-5).

- For DB2, tables must reside at the same location (subsystem). See *The DB2 Distributed Data Facility* in Appendix F, *Additional DB2 Topics*.
- The Master Files for the tables must not include any OCCURS segments.
- You must not join more than 15 tables or views.

If these conditions are satisfied, the Interface generates one SQL SELECT statement that joins all the referenced tables in a single request.

If you set Interface optimization OFF, or if any condition is not satisfied, the Interface generates an individual SQL statement for each table; FOCUS performs the required processing on the returned answer sets.

Note:

- A FOCUS TABLE request that references a dynamically joined structure generates SQL join predicates for all segments in the subtree that starts from the root segment. The Interface does not necessarily generate these predicates for multi-table Master Files (see Chapter 5, *Multi-table Structures*); in a multi-table structure the subtree effectively begins with the highest referenced segment. This effect may cause identical TABLE requests to produce different reports when run against a dynamic join structure and a multi-table Master File that represent the same tree structure.

- When the Interface does not pass a join to the RDBMS, the order in which you join the files becomes critical. To minimize the number of cursors opened, order the files from left to right by increasing number of records in either the file (for non-RDBMS files) or the returned answer set (for RDBMS tables).

  Join order should not matter in an optimized join; however, performance may degrade when you join more than three tables.

Optimization of Joins Between Heterogeneous File Types

The Interface can pass a single SELECT statement for all tables referenced in a TABLE request to the RDBMS when all active segments from the RDBMS comprise a contiguous single-path subtree. The Interface can optimize the join between relational tables even when the retrieval path includes segments from different file types (for example, several DB2 segments and an IMS segment).

Single path means no non-unique RDBMS segment can have an RDBMS parent and a non-unique sibling (RDBMS or non-RDBMS). Active segments are those containing fields referenced in the request or fields involved in join operations. Contiguous segments are connected RDBMS segments with a common target RDBMS. There may be segments of different file types above or below the contiguous segments. Segments occupying intermediate positions in a file hierarchy between explicitly active segments are themselves also implicitly active.

Note:

- If the request specifies a sort operation, the Interface transfers the sort operation to the RDBMS only if the root of the SQL subtree is the topmost active segment in the file. Additionally, all BY fields must be contained within the segments of the SQL subtree. If the multiplicative effect is detected and the Interface-managed native join is invoked, the Interface passes an SQL ORDER BY clause on all columns of each segment’s primary key, in top-to-bottom order. The resulting sort on the returned answer set enables the Interface to eliminate duplicate rows before passing the data to FOCUS.
- FOCUS and the RDBMS use slightly different sorting/aggregation algorithms. Therefore, when the RDBMS processes a sort request, the sequence of report rows within a given sort key value may vary slightly from the sequence that would be produced by FOCUS. While both report results are equally correct, if such differences are unacceptable you can set Interface optimization OFF. FOCUS will then handle all aggregation, sorts and join operations required to produce the report.

Example Optimizing Joins Between Heterogeneous File Types

The following example illustrates the SQL request passed to DB2 as the result of a FOCUS dynamic join between two DB2 tables and an IDMS record. For information about JOIN syntax, see *The Dynamic JOIN Command* in Chapter 8, *Advanced Reporting Techniques*:

```
JOIN EMP_ID IN EMPINFO TO WHO IN FUNDTRAN AS JOIN1
JOIN EMP_ID IN EMPINFO TO EMP_ID IN IDMSFILE AS JOIN2
```

With optimization enabled, the Interface produces one SQL SELECT statement that joins the two DB2 tables. The RDBMS processes the join:

```
>       SELECT T1.EID,T2.BN FROM "USER1"."EMPINFO" T1,
    "USER1"."FUNDTRAN" T2 WHERE (T2.EID = T1.EID) FOR FETCH ONLY;
```

For each DB2 joined row returned, the Interface uses the join field as input to an IDMS OBTAIN record command utilizing an IDMS index or a Calc key.

With optimization disabled, the Interface generates a separate SQL SELECT statement for each DB2 table:

```
     SELECT T1.EID FROM "USER1"."EMPINFO" T1 FOR FETCH ONLY;
```

After a row is fetched from table1, the join field is used as input to the second select:

```
     SELECT T2.BN FROM "USER1"."FUNDTRAN" T2 WHERE (T2.EID = ?) FOR
    FETCH ONLY;
```

For each DB2 joined row returned, the Interface uses the join field as input to an IDMS OBTAIN record command using an IDMS index or a Calc key.

Optimizing Sorts

Next, if the request specifies a sort operation, the Interface must determine how to sort the data. The Interface transfers the sort operation to the RDBMS only when:

- It can generate a single SQL statement (as described in *Optimizing Joins* on page 7-6).
- Interface-managed join optimization is not in effect (see *Optimizing Joins* on page 7-6).

Under these conditions, the Interface invokes one of the following types of RDBMS-managed sorting:

1. If the request does not use the direct operators FST. and LST., the Interface translates sort phrases (BY or ACROSS) into SQL ORDER BY clauses, thus passing responsibility for the primary sorting of data to the RDBMS.

2. When the Interface determines that it will retrieve at most one segment instance per sort break, it translates FOCUS FST. and LST. operators to SQL MIN and MAX operators and translates sort phrases (BY and ACROSS) into SQL ORDER BY clauses. The Interface applies this technique in requests that explicitly specify the FST. or LST. operators and in requests that implicitly use them by referencing a non-numeric field in a report heading or footing.

   FOCUS FST. and LST. operators retrieve the first or last segment instance per sort break. By definition, if a FOCUS request includes FST. or LST. operators on multiple fields from one segment, each field value displayed on the resulting report must come from the same instance of that segment. SQL MIN and MAX operators do not dictate that the resulting fields all come from the same segment instance.

   Therefore, before translating the FST. and LST. operators in a FOCUS request to SQL MIN and MAX operators, the Interface must ascertain that it will retrieve at most one segment instance per sort break. It makes this determination by analyzing the sort fields in the request, the primary key of each segment in the join structure, and any join fields that are components of the primary keys.

   One segment instance is returned per sort break in any of the following situations:

   - The sort field includes a segment's entire primary key.
   - The sort field is a partial key joined to another sort field consisting of the remaining primary key component.
   - The request includes only one FST. or LST. operator on a segment.

3. If the report request contains FST. or LST. operators on a segment that may return multiple instances, the Interface directs the RDBMS to sort the data by primary key in the sequence specified by the Access File KEYORDER attribute. From this, the Interface retrieves column values for the logically first (FST.) or last (LST.) key values of the returned data. After the RDBMS sort, FOCUS re-sorts the data according to the sort phrases.

   Note: LST. processing is invoked for SUM or WRITE operations involving alphanumeric or date fields and for alphanumeric or date fields in a report heading or footing.

From a performance standpoint, consider using the FOCUS TABLEF command in conjunction with RDBMS-managed sorting to free FOCUS from having to verify the sort order. Refer to *The TABLEF Command* in Chapter 8, *Advanced Reporting Techniques*, for the TABLEF command.

Optimizing Aggregation

FOCUS aggregation verbs SUM, COUNT, and WRITE, and direct operators MIN., MAX., and AVE., retrieve a final aggregated answer set rather than individual values. Since the RDBMS handles aggregation efficiently, the Interface structures its retrieval request so that the RDBMS performs the aggregation. The Interface passes aggregation to the RDBMS when:

- The IF or WHERE tests in the FOCUS request translate to SQL WHERE clauses. (Use FSTRACE4, described in Appendix D, *Tracing Interface Processing*, to examine the SQL generated from your request.)
- FOCUS generates a single SQL statement (as described in *Optimizing Joins* on page 7-6).
- Interface-managed join optimization is not in effect (see *Optimizing Joins* on page 7-6).
- The request specifies only the FOCUS SUM, COUNT, or WRITE aggregate verbs and the MIN., MAX., AVE., SUM., DST., and CNT. direct operators. Under certain conditions (described in *Optimizing Sorts* on page 7-9), the request can include the FOCUS FST. and LST. direct operators.
- The request does not reference DEFINE fields that do not comply with the definition of "valued expressions" (see *Valued Expressions* on page 7-15) or that are otherwise restricted (see *SQL Limitations* on page 7-16).

  Note: FOCUS calculates COMPUTE fields on its internal matrix; they do not affect the Interface's ability to pass requests for aggregation to the RDBMS.

  The Interface translates IF TOTAL and WHERE TOTAL tests to the SQL HAVING clause. It translates IF TOTAL and WHERE TOTAL tests on DEFINE and COMPUTE fields subject to the general limitations on the use of DEFINE in aggregation (see *Valued Expressions* on page 7-15).

Optimizing DEFINE Fields

The Interface can translate certain DEFINE expressions to SQL *as part of aggregation or record selection operations only*. DEFINE expressions that are not translated for the RDBMS are listed in *SQL Limitations* on page 7-16.

- For RDBMS-managed record selection, the DEFINE expression must be an arithmetic valued expression, a character string valued expression, or a logical expression.
- For RDBMS-managed aggregation, the DEFINE expression must be an arithmetic or character string valued expression.

Optimizing DEFINE Fields Referenced in FOCUS BY Clauses

The Interface can optimize aggregation requests in which a DEFINE field is the object of a FOCUS BY clause. The display commands in these requests must be FOCUS aggregation commands (SUM, COUNT, or WRITE) or direct operators such as MIN., MAX., and AVE.

- FOCUS and the RDBMS use slightly different sorting/aggregation algorithms. Therefore, when the Interface passes expressions as objects of SQL GROUP BY or ORDER BY phrases to the RDBMS, the sequence of report rows within a given sort key value may vary slightly from the sequence that would be produced by FOCUS. While both report results are equally correct, if such differences are unacceptable you can set Interface optimization OFF. FOCUS will then handle all aggregation, sorts and join operations required to produce the report.
- For DB2, this optimization enhancement applies only with DB2 version 4 and higher.

Example Passing Aggregation on DEFINE Fields to the RDBMS for Processing

This example shows the SQL passed to DB2 following aggregation on a DEFINE field that is the object of a BY clause:

```
DEFINE FILE DEDUCT
TAX = 0.085 * DED_AMT
END

TABLE FILE DEDUCT
SUM DED_AMT TAX
BY TAX NOPRINT
END

SELECT SK001, SUM(VB001), SUM(VB002) FROM (SELECT (.085 *
T1.DA) AS SK001,T1.DA AS VB001,(.085 * T1.DA) AS VB002 FROM
"USER1"."DEDUCT" T1 ) X  GROUP BY SK001 ORDER BY SK001 FOR
FETCH ONLY;
```

DB2 Interface IF-THEN-ELSE Optimization

The DB2 interface can optimize FOCUS TABLE requests that include DEFINE fields created using IF-THEN-ELSE syntax. In certain cases, such DEFINE fields can be passed to DB2 as expressions, enhancing performance and minimizing the size of the answer set returned to FOCUS.

Syntax How to Enable IF-THEN-ELSE Optimization

When you issue the DB2 Interface SET OPTIFTHENELSE command, the Interface attempts to deliver the construct of a FOCUS IF-THEN-ELSE DEFINE field to DB2 as an expression. The DEFINE field must be an object of a selection test or an aggregation request. The DEFINE definition may be specified in the TABLE request or in the Master File.

```
SQL {DB2} SET OPTIFTHENELSE {ON|OFF}
```

where:

`ON`

Enables IF-THEN-ELSE optimization.

`OFF`

Disables IF-THEN-ELSE optimization. OFF is the default.

Note: Omit the DB2 target RDBMS qualifier to issue the command if you previously issued the SET SQLENGINE command for DB2.

There is no guarantee that the SQL that is generated will improve performance for all requests. If you find that this feature does not improve performance, set OPTIFTHENELSE OFF to disable the feature.

IF-THEN-ELSE optimization applies to SELECT statements created as a result of FOCUS TABLE requests and is subject to the limitations described in *SQL Limitations* on page 7-16.

Example Using IF-THEN-ELSE Optimization Without Aggregation

Consider the following request:

```
SQL DB2 SET OPTIFTHENELSE ON
DEFINE FILE EMPINFO
DEF1 = IF (LN EQ ' ') AND (FN EQ '  ') AND (DPT EQ 'MIS') THEN 1 ELSE 0;
END

TABLE FILE EMPINFO
PRINT DPT LN FN
WHERE DEF1 EQ 1
END
```

The Interface generates an SQL request that incorporates the IF-THEN-ELSE condition corresponding to the WHERE DEF1 EQ 1 test:

```
>       SELECT T1.LN,T1.FN,T1.DPT FROM "USER1"."EMPINFO" T1 WHERE (((((T1.LN =
    ' ') AND (T1.FN = ' ')) AND (T1.DPT = 'MIS')))) FOR FETCH ONLY;
```

Example Using IF-THEN-ELSE Optimization With Aggregation

```
SQL DB2 SET OPTIFTHENELSE ON
DEFINE FILE EMPINFO
DEF2 = IF LN EQ 'SMITH' THEN 1 ELSE IF LN EQ 'JONES' THEN 2
       ELSE IF LN EQ 'CARTER' THEN 3 ELSE 0;
END

TABLE FILE EMPINFO
WRITE MAX.LN IF DEF2 EQ 1
END
```

The Interface generates an SQL request that incorporates the IF-THEN-ELSE condition corresponding to the WHERE DEF2 EQ 1 test:

```
>       SELECT  MAX(T1.LN) FROM "USER1"."EMPINFO" T1 WHERE (((T1.LN =
    'SMITH'))) FOR FETCH ONLY;
```

```
TABLE FILE EMPINFO
WRITE MAX.LN IF DEF2 EQ 2
END
```

The Interface generates an SQL request that incorporates the IF-THEN-ELSE condition corresponding to the WHERE DEF2 EQ 2 test:

```
>      SELECT  MAX(T1.LN) FROM "USER1"."EMPINFO" T1 WHERE (((NOT (T1.LN =
     'SMITH')) AND (T1.LN = 'JONES'))) FOR FETCH ONLY;
```

Example Using IF-THEN-ELSE Optimization With a Condition That Is Always False

```
SQL DB2 SET OPTIFTHENELSE ON
DEFINE FILE EMPINFO
DEF3 = IF FN EQ 'RITA' THEN 1 ELSE 0;
END

TABLE FILE EMPINFO
PRINT FN
IF DEF3 EQ 2
END
```

Because DEF3 EQ 2 will never be true, the Interface passes the WHERE test 1=0 (which is always false) to DB2, returning zero records from DB2:

```
>       SELECT T1.FN FROM "USER1"."EMPINFO" T1 WHERE (1 = 0) FOR FETCH ONLY;
```

Valued Expressions

There are two types of valued expressions: arithmetic and character string.

Arithmetic Expressions

Arithmetic expressions return a single number. DEFINE fields are classified as arithmetic if the expression on the right includes one or more of the following elements:

- Real-field operands of numeric datatype (I, P, D, F).
- Numeric constants.
- Arithmetic operators (**, *, /, + , -).
- The subtraction of one DATE field from another.
- DEFINE-field operands satisfying any of the preceding items.

For example, the arithmetic expression in the following DEFINE uses a numeric real-field operand (CURR_SAL), a previously-specified DEFINE field (OTIME_SAL), and two numeric constants (1.1 and 100):

```
NEW_SAL/D12.2= ((CURR_SAL + OTIME_SAL) * 1.1) - 100;
```

Character String Expressions

Character string expressions return a character string. DEFINE fields are classified as character strings if the expression on the right includes one or more of the following elements:

- Real-field operands of alphanumeric (A) datatype. (TX field formats are not supported in DEFINE expressions.)
- String constants.
- String concatenation operators ('|').
- DEFINE-field operands satisfying any of the preceding items.

For example, the character string expression in the following DEFINE uses two alphanumeric field operands (LAST_NAME and FIRST_NAME), a string constant (','), and string concatenation operators (|):

```
FORMAL_NAME/A36= LAST_NAME | ',' | FIRST_NAME;
```

Logical Expressions

Logical expressions return one of two values: True (1) or False (0). DEFINE fields are classified as logical if the expression on the right includes any of the following elements:

- Real-field operands of any FOCUS-supported datatype (including DATE fields).
- Constants of a datatype consistent with fields in the predicate.
- Relational operators (EQ, NE, GT, LT, GE, LE).
- Logical operators (AND, OR, NOT).
- Arithmetic or character string expression operands (described in *Valued Expressions* on page 7-15).
- DEFINE-field operands satisfying any combination of the preceding items.

For example, the logical expression in the following DEFINE is composed of two expressions connected by the logical operator OR; each part is itself a logical expression:

```
SALES_FLAG/I1= (DIV_CODE EQ 'SALES') OR (COMMISSION GT 0);
```

In the next example, the DEFINE field, QUOTA_CLUB, is the value of a logical expression composed of two other expressions connected by the logical operator AND. Note that the first expression is the previously-specified DEFINE field, SALES_FLAG:

```
QUOTA_CLUB/I1= (SALES_FLAG) AND (UNITS_SOLD GT 100);
```

SQL Limitations

Since the FOCUS reporting language is more extensive than native SQL, the Interface cannot pass certain DEFINE expressions to the RDBMS for processing. The Interface does not offload DEFINE-based aggregation and record selection if the DEFINE includes:

- User-written subroutines.
- Self-referential expressions such as:

  ```
  X=X+1;
  ```
- EDIT functions for numeric-to-alpha or alpha-to-numeric field conversions.
- DECODE functions for field value conversions.
- Relational operators CONTAINS, OMITS, IS-MISSING, INCLUDES, and EXCLUDES.

- FOCUS subroutines ABS, INT, MAX, MIN, LOG, and SQRT.

  Note: Do not confuse the FOCUS user-written subroutines MAX and MIN with the MAX. and MIN. prefix operators. DEFINE fields cannot include prefix operators.

- Expressions involving fields with ACTUAL=DATE, except for the subtraction of one DATE field from another and all logical expressions on DATE fields.
- Extended Matrix Reporting (EMR) cell calculations. EMR is also referred to as Financial Reporting Language (FRL).

  Note: EMR report requests are extended TABLE requests. The Extended Matrix Reporting Language provides special functions for detailed reporting; consult the *FOCUS for IBM Mainframe User's Manual* for more information.

In addition, DB2 Interface IF-THEN-ELSE optimization does not support the following features:

- Any type of DECODE expression.
- STATIC SQL.
- IF/WHERE DDNAME.
- Partial Date selection.

The FOCUS EXPLAIN Utility

The FOCUS for SQL EXPLAIN utility is an interactive development tool that helps you fine-tune FOCUS query performance. It invokes the RDBMS EXPLAIN function to analyze the efficiency of data retrieval paths for TABLE requests; you cannot use the EXPLAIN utility with MODIFY, MAINTAIN, or MATCH FILE requests. The analysis result is displayed in the FOCUS Hot Screen facility; you can save or print it for further examination.

This section provides:

- An overview of internal processing in *Processing Overview* on page 7-17.
- Instructions for invoking the FOCUS EXPLAIN utility and descriptions of its prompting windows in *Using the EXPLAIN Utility* on page 7-18.
- A sample report request and its EXPLAIN result in *Sample EXPLAIN Report* on page 7-22.

Processing Overview

Given a TABLE request, the EXPLAIN utility executes the Interface and generates SQL statements using its normal mechanisms, the FOCUS TABLE parser and Dialogue Manager.

However, instead of processing the request, FOCUS directs the RDBMS to invoke its own native EXPLAIN function and analyze the generated statements. The analysis produces a detailed evaluation of the **access path**, the methodology for retrieving the data for that request.

The RDBMS places this access path information into a table (DB2) or set of tables (SQL/DS). The FOCUS EXPLAIN utility reads these tables and generates a clear and detailed report containing valuable information about the performance characteristics of your query, information that anyone familiar with the RDBMS and its performance characteristics can understand and analyze.

Using the EXPLAIN Utility

Enter FOCUS and execute the EXPLAIN utility with the following syntax

```
EX {EXPDB2|EXPSQL}
```

where:

`EXPDB2`
: Is the FOCEXEC that invokes the DB2 EXPLAIN function.

`EXPSQL`
: Is the FOCEXEC that invokes the SQL/DS EXPLAIN function.

Press the Enter key.

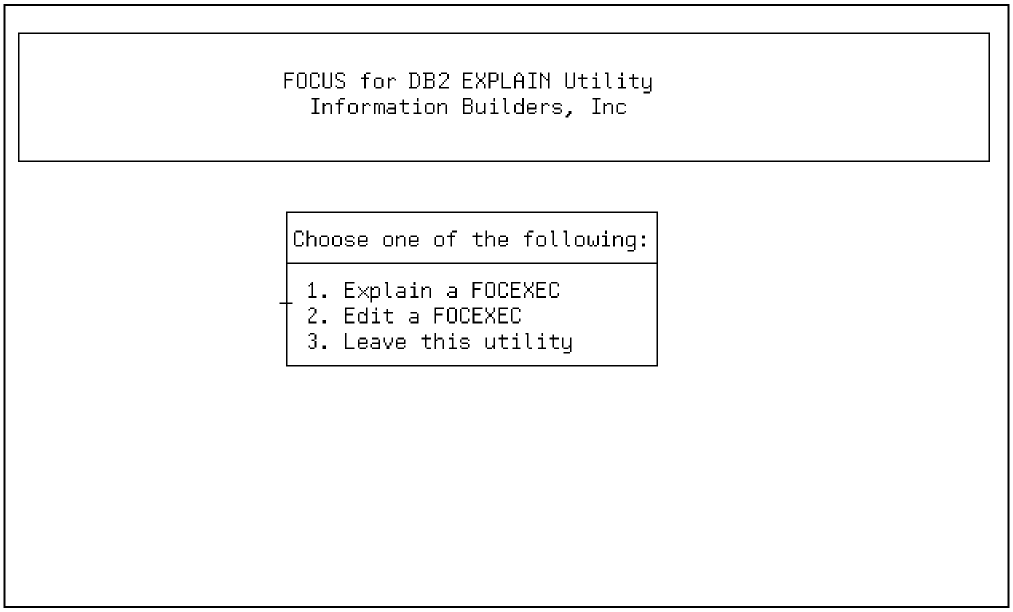
Note:

- New versions of the EXPLAIN utility are available for DB2 Version 3 and SQL/DS Version 3 Release 4. (On the distribution tape, these new versions are called EXPDB231 for DB2 and EXPSQL34 for SQL/DS.) Ask your system support staff for the correct FOCEXEC name to use at your site.
- If the IM parameter was set to zero when the Interface was installed, you do not have access to the EXPLAIN utility; the Interface will generate error message FOC1638 if you attempt to use it. See the *FOCUS for IBM Mainframe DB2 and SQL/DS Read/Write Interface Installation Guide* for more information.

The EXPLAIN utility cannot analyze an interactive TABLE request; you must provide the name of a FOCEXEC on the main window. However, you can access the TED editor from within the FOCUS EXPLAIN utility and create or change the TABLE request at will.

If you do not have the tables required for executing the RDBMS EXPLAIN function, the FOCUS EXPLAIN utility attempts to create them. In DB2, you need appropriate authorization for creating tables, otherwise an SQLCODE of -551 results. The report results from the FOCUS EXPLAIN utility are displayed in Hot Screen; you can save or print them for later examination.

The main window presents three choices:



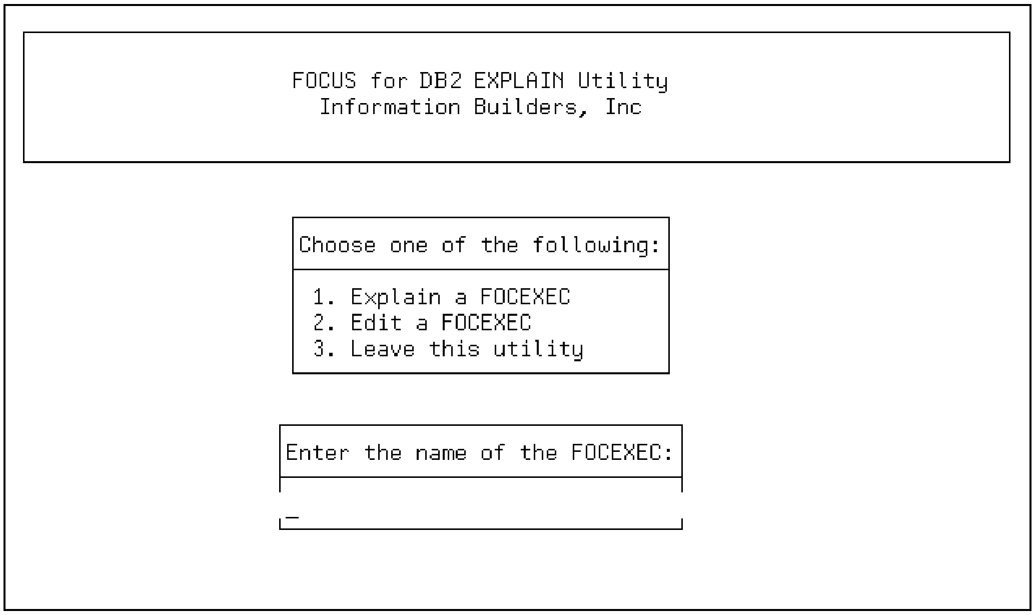
```
FOCUS for DB2 EXPLAIN Utility
Information Builders, Inc

Choose one of the following:

1. Explain a FOCEXEC
2. Edit a FOCEXEC
3. Leave this utility
```

To make a selection, move your cursor under one of the numbers and press the Enter key. To exit the utility and return to the FOCUS command line, press the PF3 key. The choices are:

1.	Explain a FOCEXEC	Invokes the RDBMS EXPLAIN command for the TABLE request in your FOCEXEC.
2.	Edit a FOCEXEC	Invokes the TED editor from within the FOCUS EXPLAIN utility. You can create a new FOCEXEC or edit an existing one.
3.	Leave this utility	Returns you to FOCUS, deleting any entries made in the RDBMS EXPLAIN tables.

At any point, you can use the PF3 key to return to a previous window.

If you select Choice 1 or 2, you are asked for the name of your FOCEXEC:



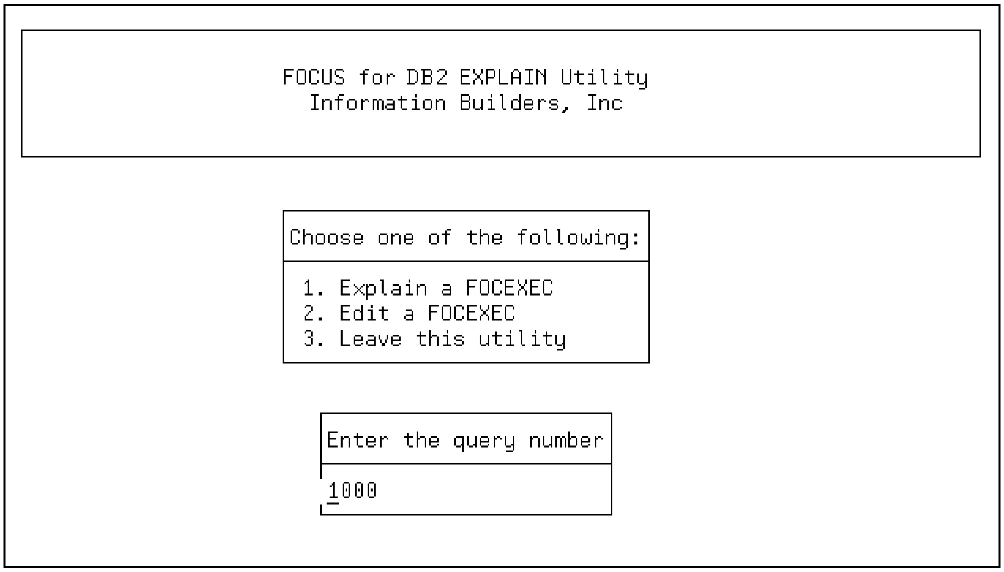
```
FOCUS for DB2 EXPLAIN Utility
Information Builders, Inc

Choose one of the following:

1. Explain a FOCEXEC
2. Edit a FOCEXEC
3. Leave this utility

Enter the name of the FOCEXEC:
_
```

If you are creating a new FOCEXEC, the same FOCEXEC naming conventions apply here as for the FOCUS EXECUTE (EX) command. Specify the TSO member name or the CMS filename. If you have already used TED (Choice 2), the name of the most recent FOCEXEC is automatically supplied.

Press the Enter key and continue with either the EXPLAIN option or the TED editor:

- The EXPLAIN option. The option to explain a FOCEXEC requires a query number, an internal control number used by the RDBMS when populating the EXPLAIN tables. You can choose any number between 1 and 32,767. However, if you choose a number that already exists, the EXPLAIN utility informs you of its existence and deletes all entries with that number before processing your request. Therefore, make sure the number you choose does not correspond to an existing entry that you would like to keep. If there are no existing entries in the EXPLAIN tables, any number will do.

For example, the query number 1000 has been entered on the following screen:



```
FOCUS for DB2 EXPLAIN Utility
 Information Builders, Inc

Choose one of the following:

 1. Explain a FOCEXEC
 2. Edit a FOCEXEC
 3. Leave this utility

Enter the query number

1000
```

At this point, the message “PLEASE WAIT, PROCESSING YOUR REQUEST” appears, followed by your EXPLAIN report.

Your EXPLAIN report is displayed via the Hot Screen facility. All Hot Screen options are available. When you exit Hot Screen, you return to the first window and may evaluate another request.

- The TED editor. Edit a new or existing FOCEXEC. When you are finished, type FILE (to save your edits) or QUIT. Do not issue the RUN command from TED within the EXPLAIN utility. You return to the initial main window and can evaluate your FOCEXEC with Choice 1, the EXPLAIN option.

Note:

- With the EXPLAIN option (Choice 1), if you:
    - Specify a FOCEXEC that generates FOCUS or SQL errors, the EXPLAIN utility stops processing and you return to the utility main menu.
    - Misspell the name of a FOCEXEC or specify one that does not exist, the EXPLAIN utility stops processing and you return to the FOCUS command prompt.
- Do not execute large Dialogue Manager applications through the EXPLAIN utility; the FOCUS TABLE parser executes the Dialogue Manager statements and may produce unpredictable results.

- Create separate FOCEXECs for evaluating individual report requests. Experiment with alternate parameters to improve response time and RDBMS performance.
- Do not use the FSTRACE4 trace in any FOCEXEC that you evaluate with the EXPLAIN utility.

Sample EXPLAIN Report

FOCEXEC RPT1 contains the following TABLE request:

```
TABLE FILE INVQ5
SUM PRICE BY PARTNO
WHERE DESCRIPTION EQ 'BOLT' OR 'NUT' OR 'SCREW'
WHERE PRICE GT .30
IF TOTAL PRICE GT 1
END
```

The EXPLAIN output consists of two reports. The first is a one-page report:

```
PAGE     1

        EXPLAIN REPORT 1 FOR FOCEXEC RPT1                    RUN ON 06/01/95
                                               QUERY NUMBER IS 1000
                  THE FOLLOWING SQL STATEMENT(S) WILL BE EXPLAINED


       SELECT T1.PARTNO, SUM(T2.PRICE) FROM "TESTID"."INVENT5" T1,
      "TESTID"."QUOT5" T2 WHERE (T2.PARTNO = T1.PARTNO) AND
      (T1.DESCRIPTION IN('BOLT', 'NUT', 'SCREW')) AND (T2.PRICE > .3)
      GROUP BY T1.PARTNO HAVING ( SUM(T2.PRICE) > 1.) ORDER BY
      T1.PARTNO FOR FETCH ONLY;













                                                                        MORE
```

The second is a four-page report:

```
  PAGE     1

    EXPLAIN REPORT 2 FOR FOCEXEC RPT1                    RUN ON 06/01/95
                                            QUERY NUMBER IS 1000


   1ST   ACCESS OF DATA

  TABLE NAME ........................:  TESTID.INVENT5
  TABLE NUMBER ......................:  1
  JOIN METHOD .......................:  FIRST TABLE ACCESSED
  MULTIPLE INDEX OPERATION SEQUENCE .:  0
  INDEX ACCESS TYPE .................:  DIRECT INDEX ACCESS
  # OF INDEX KEYS USED ..............:  0
  INDEX NAME ........................:  TESTID.INVENT5IX
  INDEX ONLY ACCESS? ................:  NO
  SORT OF BASE TABLE REQUIRED FOR ...:  NOTHING
  SORT OF RESULT TABLE REQUIRED FOR .:  NOTHING
  LOCKING MODE ......................:  INTENT SHARE
  TYPE OF PREFETCH ..................:  UNKNOWN/NO PREFETCH
  COLUMN FUNCTION EVALUATED AT ......:  N/A OR DETERMINED AT EXECUTION


                                                                      MORE
```

```
PAGE     2

    EXPLAIN REPORT 2 FOR FOCEXEC RPT1                    RUN ON 06/01/95
                                            QUERY NUMBER IS 1000

  PARALLEL ACCESS DEGREE ............:  .
  PARALLEL ACCESS GROUP .............:  .
  PARALLEL JOIN DEGREE ..............:  .
  PARALLEL JOIN GROUP ...............:  .









                                                                      MORE
  PAGE     3

    EXPLAIN REPORT 2 FOR FOCEXEC RPT1                    RUN ON 06/01/95
                                         QUERY NUMBER IS 1000



   2ND   ACCESS OF DATA

  TABLE NAME .........................:  TESTID.QUOT5
  TABLE NUMBER .......................:  2
  JOIN METHOD ........................:  NESTED LOOP JOIN
  MULTIPLE INDEX OPERATION SEQUENCE .:  0
  INDEX ACCESS TYPE ..................:  DIRECT INDEX ACCESS
  # OF INDEX KEYS USED ...............:  0
  INDEX NAME .........................:  TESTID.QUOT5IX
  INDEX ONLY ACCESS? .................:  NO
  SORT OF BASE TABLE REQUIRED FOR ....:  NOTHING
  SORT OF RESULT TABLE REQUIRED FOR .:  NOTHING
  LOCKING MODE .......................:  INTENT SHARE
  TYPE OF PREFETCH ...................:  UNKNOWN/NO PREFETCH
  COLUMN FUNCTION EVALUATED AT .......:  N/A OR DETERMINED AT EXECUTION



                                                                     MORE
```

```
 PAGE     4

    EXPLAIN REPORT 2 FOR FOCEXEC RPT1                    RUN ON 06/01/95
                                         QUERY NUMBER IS 1000

  PARALLEL ACCESS DEGREE .............:  .
  PARALLEL ACCESS GROUP ..............:  .
  PARALLEL JOIN DEGREE ...............:  .
  PARALLEL JOIN GROUP ................:  .

  IF YOUR REQUEST SPONSORED A SEQUENTIAL SCAN OF THE DATA, OR
  ONE OR MORE ADDITIONAL SORTS, ESPECIALLY ON THE COMPOSITE
  RESULT TABLES, YOU MAY WISH TO REPHRASE THIS REPORT











                                  END OF REPORT
```

For information about the EXPLAIN report, consult the *DB2 Administration Guide*.

8 Advanced Reporting Techniques

Topics:

- The TABLEF Command
- Creating Extract Files on the RDBMS
- The Dynamic JOIN Command
- Missing Rows of Data in Cross-referenced Tables
- JOIN Utilities
- Search Limits
- Multiple Retrieval Paths

Using the Interface, you can create graphs with FOCUS GRAPH requests, design interactive procedures with Dialogue Manager, or extract data from reports with the Hot Screen Facility. However, of all the facilities the Interface makes available to you for reading, retrieving, and organizing data from RDBMS tables, the most widely used is the Report Writer, invoked by the FOCUS TABLE, TABLEF, and MATCH commands. Some reporting features for the Interface vary from standard FOCUS; these minor variations are discussed in this chapter.

This chapter includes:

- Improving retrieval performance with the TABLEF command. (See *The TABLEF Command* on page 8-2.)
- Creating RDBMS tables from a TABLE request. (See *Creating Extract Files on the RDBMS* on page 8-3.)
- Joined structures for reporting purposes. How multi-table Master Files compare with dynamically-joined structures, the JOIN command for unique and non-unique relationships, and the multi-field JOIN command. (See *The Dynamic JOIN Command* on page 8-9.)
- The SET ALL command and multi-table structures. How short retrieval paths affect report results. (See *Missing Rows of Data in Cross-referenced Tables* on page 8-17.)
- The join utilities CHECK FILE, ? JOIN, and JOIN CLEAR. (See *JOIN Utilities* on page 8-29.)
- Specifying search limits with the READLIMIT and RECORDLIMIT commands. (See *Search Limits* on page 8-31.)

- For multi-table structures only, multiple retrieval paths and how sort phrases and screening statements affect retrieval. (See *Multiple Retrieval Paths* on page 8-33.)

If you are not familiar with FOCUS syntax, you can create report requests with TableTalk, a menu-driven report generator. Also, consult the *FOCUS Report Writing Primer* for more information about FOCUS reporting.

Some FOCUS and SQL query functions are similar:

- SQL SELECT is equivalent to the FOCUS verb PRINT. Both display individual values for a given column or field.
- SQL aggregate functions SUM, COUNT *, MAX, MIN, and AVG are comparable to FOCUS SUM., CNT., MAX., MIN., and AVE. field prefix operators.
- The SQL WHERE predicate is similar to FOCUS IF or WHERE screening tests.
- The SQL ORDER BY sort phrase is equivalent to the FOCUS BY sort phrase.
- SQL set operations such as UNION, intersection, and difference are available as FOCUS MATCH FILE options. The FOCUS MATCH FILE command generates a subset of data from tables or views. The subset is stored in a HOLD extract file and is available as a source for further report processing. Refer to the *FOCUS for IBM Mainframe User's Manual* for more information about the MATCH FILE command.

  You can issue an SQL SELECT request from within FOCUS using the Direct SQL Passthru facility (see Chapter 9, *Direct SQL Passthru*).

The TABLEF Command

TABLEF FILE is an alternate for the TABLE FILE command when the answer set returned from the RDBMS is already in the sort order required for the report. TABLEF is faster and more efficient because FOCUS does not build (and sort) an internal matrix, but works directly on the data returned by the RDBMS.

To further improve efficiency, create an SQL clustered index on the sort field used in the TABLEF request. The RDBMS physically stores the data in the same order as the clustered index, so the request requires fewer I/Os to data pages.

Most reporting syntax is acceptable in TABLEF requests, but be aware of the following:

- If BY phrases are passed to the RDBMS as SQL ORDER BY phrases, the answer sets returned are sorted and FOCUS does not have to sort them again or verify the sort order.

  In some circumstances FOCUS does not pass an ORDER BY phrase to the RDBMS (see Chapter 7, *The Interface Optimizer*), so you must verify that the ORDER BY is passed before relying on the sorted order of the returned answer set. Use FSTRACE4 (see Appendix D, *Tracing Interface Processing*) to examine the SQL statements generated by the Interface.

- ACROSS phrases are not supported.
- Multi-verb requests are not supported.
- The RETYPE command is not available.
- You can include the ON TABLE HOLD statement in the TABLEF request to produce an extract file.

Note: In a request that generates report output to the terminal, TABLEF holds locks on data pages until the Interface issues a COMMIT (usually when the report display is terminated). If the Isolation Level (see *Isolation Levels* in Chapter 12, *Maintaining Tables With FOCUS*) is set to Repeatable Read (RR), all pages accessed to produce the report are locked; if it is set to Cursor Stability (CS), the last page accessed is locked. These locks may prevent access to the data by other applications; if this presents a problem, reports generated to the terminal using TABLEF should be processed and terminated as quickly as is feasible. This consideration does not apply to regular TABLE requests, because the Interface normally issues the COMMIT prior to displaying the report. Nor is it a problem for requests that generate only offline reports or extract (HOLD) files, as the answer set is exhausted automatically.

Creating Extract Files on the RDBMS

Using FOCUS TABLE syntax, you can create extract files (tables) in the RDBMS. You can then use these tables, like any other RDBMS table, for both read-only and read-write operations. In fact, with the FOCUS HOLD FORMAT SQL or DB2 option, you can create RDBMS tables from *any* FOCUS-readable file. This feature facilitates data migration and leaves the original source unaffected.

In order to create RDBMS tables and indexes, you must have an adequate level of RDBMS authority. Your site must also have enabled WRITE access and native SQL support when installing the Interface. Contact your site DBA for more information.

To extract data and convert it to an RDBMS table, issue the HOLD command with the FORMAT SQL or DB2 option either in the report request or after the report has printed. The Interface generates a single-table Master and Access File, and it creates and loads one RDBMS table. If the report request uses the verbs PRINT or LIST, it also creates both a FOCLIST field with internal list values and a unique index on all BY fields and the FOCLIST field. If the request uses the verb SUM, the Interface creates a unique index on any BY fields.

If you attempt to issue the HOLD FORMAT SQL or DB2 command without first installing the Interface, the following error messages will be generated:

```
(FOC1488)    SQL INTERFACE IS NOT INSTALLED
(FOC1479)    ERROR CONNECTING TO SQL DATABASE
```

Note: Chapter 11, *Environmental Commands*, describes how to control index space parameters with the Interface SET IXSPACE command.

The RDBMS table name that results from the extract must not already exist in the DB2 subsystem or SQL/DS database. There must be a tablespace in DB2 or dbspace in SQL/DS to receive the newly created table. Use the SET DBSPACE command to specify one. You can view the default tablespace or dbspace setting with the SQL ? command. (See Chapter 11, *Environmental Commands*, for a discussion of Interface environmental commands.)

Within the report request, the syntax is:

```
ON TABLE HOLD [ AS name ] FORMAT {DB2|SQL}
```

At the command level, the syntax is

```
HOLD [ AS name ] FORMAT {DB2|SQL}
```

where:

name
: Is a name for the extract file; the default name is HOLD. Maximum 27 characters (including a period to separate the creator ID from the table name). The first 8 characters become the resulting file description name and the FILENAME value in the Master File. All 27 characters become the TABLENAME value in the Access File.

  Note: If the name contains a period (.), the characters preceding it are treated as the creator ID; characters following the period become the file description name and the FILENAME value in the Master File. Consult Chapter 4, *Describing Tables to FOCUS*, for information about creator IDs and TABLENAME values.

SQL|DB2
: Indicates that the HOLD extract data file is stored as a table in either DB2 or SQL/DS. DB2 is a synonym for SQL.

Unless you give them an AS name, file descriptions are temporary and exist only for the current session. All HOLD tables are permanent and must be explicitly dropped.

To make the HOLD file descriptions permanent, specify an AS name in the HOLD command, and consider the following:

- In DB2, the generated Master and Access Files are created as members of the partitioned data sets allocated to DDNAMEs HOLDMAST and HOLDACC. If you do not allocate those DDNAMEs, they are allocated dynamically and deleted at the end of the session. To permanently retain the new file descriptions, allocate HOLDMAST to a permanent partitioned data set and HOLDACC to a second permanent partitioned data set.
- In SQL/DS, by default the generated Master and Access Files are named HOLD MASTER and HOLD FOCSQL. They reside on the A disk by default, any writable disk specified with the FOCUS SET TEMP command, or on the writable disk with the most space.

Example Converting the FOCUS PROD Database to an RDBMS Table

This example converts the FOCUS PROD database to a table:

```
sql db2 set dbspace public.space0
>
table file prod
print *
on table hold as prodsql format sql
end

    NUMBER OF RECORDS IN TABLE=       14  LINES=     14


    HOLDING SQLDS    FILE...
```

Generated Master Files

Even if the original Master File describes several segments, the Master File resulting from the ON TABLE HOLD extract is a single-table description; it contains the attributes described in *Master Files* in Chapter 4, *Describing Tables to FOCUS*. Following is a list of the generated keyword/value pairs:

- The default FILENAME value is HOLD. If you specified a name with the AS phrase, the FILENAME value is the first eight characters of that name. If the name contains a period (.), the characters preceding the period are treated as the creator id; characters following the period become the resulting file description name and FILENAME value in the Master File.
- The SUFFIX value is SQLDS.
- The SEGNAME value is SEG01 and the SEGTYPE value is S0.
- FIELDNAME and ALIAS values from the original Master File are retained. If the original Master File does not define aliases, the original fieldnames are also the new ALIAS values.

  If the report request contains an AS phrase to rename a field, the AS phrase name becomes the new value for FIELDNAME and the original fieldname becomes the new value for ALIAS. The AS phrase name is also included as a TITLE value.

  The FOCUS verbs PRINT and LIST create an additional field named FOCLIST that contains internal list values.
- The resulting USAGE field formats are generally the same as the original USAGE formats. The new ACTUAL formats are converted based on original USAGE formats and certain conditions (see the chart in *Extract File Conversion Chart* on page 8-8). If the original Master File contains ACTUAL formats, they are ignored.
- MISSING parameter settings are converted to SQL NULL statements.

Example Sample Generated Master File

The following example is the generated Master File from the ON TABLE HOLD AS PRODSQL statement in *Converting the FOCUS PROD Database to an RDBMS Table* on page 8-5:

```
FILE=PRODSQL        ,SUFFIX=SQLDS,$
SEGNAME=SEG01   ,SEGTYPE=S0,$
FIELDNAME   =FOCLIST      ,FOCLIST     ,I5       ,I4    ,$
FIELDNAME   =PROD_CODE    ,PCODE       ,A3       ,A3    ,$
FIELDNAME   =PROD_NAME    ,ITEM        ,A15      ,A15   ,$
FIELDNAME   =PACKAGE      ,SIZE        ,A12      ,A12   ,$
FIELDNAME   =UNIT_COST    ,COST        ,D5.2M    ,D8    ,$
```

Generated Access Files

The Access File resulting from the ON TABLE HOLD extract contains the declarations described in *Access Files* in Chapter 4, *Describing Tables to FOCUS*. For PRINT and LIST based reports, the FOCLIST field and the BY phrases determine the KEYS value and how the index is created. Following is a list of the generated keyword/value pairs:

- The SEGNAME value is SEG01.
- The TABLENAME value may be up to 27 characters long (including a period) and defaults to HOLD. If you specified a name with the AS phrase, that name becomes the TABLENAME value. If the name contains a period (.), the characters preceding it are considered the creator name; characters following the period become the table name.

  Note: If you do not specify a creator name, the ID specified by the DB2 SET OWNERID command becomes the creator. If you did not issue this command, your userid becomes the creator by default.
- The KEYS value and the fields that are indexed depend on the verb and whether BY sort phrases are used in the request:
  - If the request specifies the NOPRINT option for a BY phrase, that sort field is not included in the table or the calculation of the KEYS value, and it is not indexed.
  - All PRINT (or LIST) requests generate a FOCLIST field. SUM (or COUNT) requests do not.
  - If the request specifies an aggregate verb such as SUM (or COUNT) with printed BY phrases, the KEYS value equals the number of printed sort fields.
  - If the request specifies the verb PRINT (or LIST) with printed BY phrases, the KEYS value equals the number of printed sort fields plus one for the generated FOCLIST field.
  - If there are no BY phrases in the request, the KEYS value is 01 regardless of the verb.

- The Interface creates a unique index on the printed BY fields plus the FOCLIST field (if it exists). If there are no BY fields, for a PRINT (or LIST) request, the Interface creates a unique index on FOCLIST alone; for a SUM (or COUNT) request, it creates a unique index on the first column referenced in the request.
- The WRITE value is YES.

Example Sample Generated Access File

For example, the generated Access File from the ON TABLE HOLD AS PRODSQL statement in *Converting the FOCUS PROD Database to an RDBMS Table* on page 8-5:

```
SEGNAME=SEG01   ,
TABLENAME=PRODSQL  ,
KEYS=01  , WRITE=YES, $
```

Other Generated Files

Three work files, FOC$HOLD MASTER, FOC$HOLD FOCTEMP, and FOCSORT, are used with an internal MODIFY procedure to create and load the extract table. The FOC$HOLD Master File is a fixed-format file with a corresponding sequential data file.

Usage Restrictions

The following restrictions apply to the HOLD FORMAT DB2 and SQL options:

- You may not use the HOLD FORMAT DB2 or SQL option in multi-verb TABLE requests; only one verb is allowed.
- You may not use the ACROSS sort phrase.
- Names used in AS phrases to rename fields should begin with an alpha character. If the name begins with a digit, SQL error -105 results and the table is not generated. For more information on AS names, refer to the *FOCUS for IBM Mainframe User's Manual*.

Extract File Conversion Chart

The following chart shows original USAGE formats, conditions, and the resulting USAGE and ACTUAL formats for ON TABLE HOLD; the field length is represented by n:

Non-SQL USAGE	**Conditions**	**HOLD USAGE**	**HOLD ACTUAL**
An	none	An	An
Dn	none	Dn	D8
Fn	none	Fn	F4
In	n EQ 1 or 2 n GT 2 MISSING=ON	In In In	I2 I4 I4
Pn	n LE 31 MISSING=ON	Pn Pn, n≤15	P(trunc((n+2)/2)) P8 **Note:** m=1 if the original source file USAGE format contains a decimal point, m=0 otherwise.
date	Date format	date	DATE
TXnn	TEXT field	TXnn	TX

Note:

- USAGE field types A, D, F, and TX from the original Master File remain the same; their generated ACTUAL formats vary slightly.
- USAGE field types I and P from the original Master File are converted based on the original length and on whether null values are permitted (MISSING=ON).
- USAGE values for FOCUS date formats remain the same; they are converted to the ACTUAL format DATE.
- To create SMALLINT NOT NULL columns with the CREATE FILE command, HOLD FORMAT DB2, or HOLD FORMAT SQL, use an ACTUAL attribute of I2, and then change the ACTUAL attribute to I4. In all other cases, use I4 for the ACTUAL attribute.

The Dynamic JOIN Command

With the FOCUS dynamic JOIN command, you can reference two or more related tables (or external files) in a single FOCUS report request. The data structures remain physically separate, but FOCUS treats them as a single logical structure. The terms **primary key** and **foreign key** refer to the common columns that relate host and cross-referenced tables.

Although the run-time effect of the dynamic join is very similar to that of the embedded join discussed in Chapter 5, *Multi-Table Structures*, the dynamic join is easier to construct since it does not require a separate Master File and Access File. You can create a dynamic join on an as-needed basis.

Note: A FOCUS TABLE request that references a dynamically joined structure generates SQL join predicates for all segments in the subtree that starts from the root segment. Multi-table Master Files do not necessarily generate these predicates; in a multi-table structure, the subtree effectively begins with the highest referenced segment. This difference may cause identical TABLE requests to produce different reports when run against a dynamic join structure and a multi-table Master File that represent the same tree structure.

The dynamic join is limited to FOCUS read-only operations (for example, TABLE, GRAPH, and the MODIFY LOOKUP facility). Embedded joins in a Master File support both read-only and read-write operations.

FOCUS provides two types of dynamic join. The difference between the two depends on the cross-referenced relation and its foreign key values:

- The multiple or non-unique join defines a one-to-many or many-to-many correlation between the records of the host file and the records of the cross-referenced file. That is, for any row in the host file, there may be more than one corresponding row in the cross-referenced file.
- The unique join defines a one-to-one correlation between a record in the host file and one record in the cross-referenced file. For any row in the host file, there is, at most, one corresponding row in the cross-referenced file. When FOCUS executes a unique join, it retrieves only one row from the cross-referenced file; be careful not to identify a join as unique to FOCUS if it is really non-unique.

  The unique join is a FOCUS concept. The RDBMS makes no distinction between unique and non-unique situations; it always retrieves all matching rows from the cross-referenced file.

  Note: If the RDBMS processes a join that the FOCUS request specifies as unique, and if there are, in fact, multiple corresponding rows in the cross-referenced file, the RDBMS inner join returns all matching rows. If, instead, optimization is disabled so that FOCUS processes the join, a different report results because FOCUS, respecting the unique join concept, returns only one cross-referenced row for each host row.

With either type of join, some rows in the host table may lack corresponding rows in the cross-referenced table. In a report request, such a retrieval path is called a *short path*.

When a report omits host rows that lack corresponding cross-referenced rows, the join is called an *inner join*. When a report displays all matching rows plus all rows from the host file that lack corresponding cross-referenced rows, the join is called a *left outer join*.

Sometimes FOCUS passes responsibility for a join to the RDBMS. The OPTIMIZATION setting is one factor in determining whether a join is optimized; other factors depend on the specific elements in the report request (see Chapter 7, *The Interface Optimizer*). In order to understand join behavior, you must know whether optimization is enabled or disabled for a particular report request. FSTRACE3 gives you that information (see Appendix D, *Tracing Interface Processing*).

For both embedded and dynamic joins, factors that determine whether a report includes short path rows are the type of join, the FOCUS SET ALL command, and whether FOCUS or the RDBMS is handling the join. Subsequent sections describe how these factors interact. SET ALL is described in *The SET ALL Command* on page 8-18.

Single-field Dynamic Join

The syntax for a join based on relating one column of the host and cross-referenced tables is

```
JOIN fielda1 [WITH rfield] IN hostfile [TAG tag1] TO [ALL]
fieldb1 IN crfile [TAG tag2] AS joinname
[END]
```

where:

fielda1

Is a field in the host file or a DEFINE field that shares values and format with fieldb1 in the cross-referenced file.

WITH *rfield*

Use only if fielda1 is a DEFINE field; associates the DEFINE field with the segment location of a real field (rfield) in the host file.

hostfile

Is the name of the host file. Use this name in subsequent TABLE requests on the joined structure.

tag1

Is the tag name for the host file, a one- to eight-character table name qualifier for field and alias names in the host file. The tag name for the host must be the same in all the JOIN commands for a join structure. The tag name may not be the same as any table name in the structure.

`ALL`

Indicates that the host and cross-referenced files participate in a one-to-many or many-to-many relationship. That is, for any value of the join field in the host file (fielda1), there may be more than one corresponding instance of that value for the join field in the cross-referenced file (fieldb1). In FOCUS terminology, this is known as a non-unique or multiple join.

Note: The use of the ALL parameter does not disable optimization. Do not confuse this ALL parameter with the SET ALL command discussed in *The SET ALL Command* on page 8-18.

Omitting the ALL parameter indicates a unique join. Omit the ALL parameter only when each row in the host file has, at most, one corresponding row in the cross-referenced file. The unique join is a FOCUS concept; the RDBMS makes no distinction between unique and non-unique situations when it processes a join.

`fieldb1`

Is a field in the cross-referenced file whose values and format can match that of fielda1.

`crfile`

Is the name of the cross-referenced file.

`tag2`

Is the tag name for the cross-referenced file, a one- to eight-character table name qualifier for field and alias names in the cross-referenced file. The tag name may not be the same as any table name in the structure. In a recursive join structure, if no tag name is provided, FOCUS prefixes all fieldnames and aliases with the first four characters of the join name.

`joinname`

Assigns an internal name to the join structure, up to eight characters in length. Joinname also provides a unique prefix for fieldnames participating in a recursive join.

You can clear the join name when you no longer need this join (see *JOIN CLEAR* on page 8-31). Do not specify joinname as the filename in subsequent TABLE FILE requests; use hostfile.

`END`

Required when the JOIN statement is longer than one line; terminates the statement.

The following is an example of a single-field dynamic join:

```
JOIN EID IN EMPINFO TAG FILE1 TO ALL PAYEID IN PAYINFO TAG FILE2 AS JOIN1
```

You can join up to 16 structures to create a FOCUS view of the data. The joined structure remains in effect for the duration of the FOCUS session or until you clear the join name using the JOIN CLEAR command (see *JOIN CLEAR* on page 8-31). The Interface instructs the RDBMS to perform a join based on the smallest subtree of referenced tables (from the root) required to satisfy the request.

Note: For a join to be optimized, it cannot involve more than 15 tables or views, the RDBMS limit for an SQL statement.

Each cross-referenced table must have at least one data field in common with its host file. Fixed sequential, ISAM, VSAM, IMS, CA-IDMS/DB, CA-Datacom/DB, ADABAS, Teradata, Model 204, Oracle, and RDBMS tables and views can be both host files and cross-referenced files in any combination. You can also join these files to all segments of a FOCUS database by using FOCUS database indexed fields.

If the DB2 Distributed Data Facility is installed:

- You can use the dynamic JOIN command to join tables from two different DB2 subsystems. FOCUS processes the join since DB2 does not allow a single SQL statement to reference tables at more than one location.
- When it detects the implicit or explicit presence of multiple LOCATION attributes in the Access File for the tables referenced in the report request, the Interface temporarily disables optimization so that FOCUS can manage the join. If you are using local RDBMS aliases for remote tables, you must set OPTIMIZATION OFF for each request that joins tables from more than one location.

The following chart lists some of the more common combinations available with the FOCUS dynamic JOIN command and restrictions on their use. Consult the *FOCUS for IBM Mainframe User's Manual* for additional file combinations and examples of dynamic joins. In the chart, SQL refers to DB2, SQL/DS, Teradata, or Oracle tables and views:

From	**To**	**Special Rules That May Apply**
SQL	SQL	Joined fields must be of the same datatype. For efficient retrieval, their lengths should also be equivalent. Use of indexed fields for the host and cross-referenced fields is recommended, but the RDBMS uses the indexes only if both the datatypes and lengths are equal.
SQL	FOCUS	Must join to an indexed field (FIELDTYPE=I). Joined fields must have common datatype and length.
SQL	IMS	XMI/DLI, XMI/MVS/IMS, or IMS Interface must be installed. The DL/I join field must be a key field in the root segment of the database; it can be a primary or secondary index. Join fields must have a common datatype and length.
SQL	VSAM	For a unique join, the VSAM join field must be a full primary key. For a non-unique join, the join field can be an initial subset of the primary key. Joined fields must have a common datatype and length.
SQL	QSAM	QSAM file must be sorted by its join field. Joined fields must have a common datatype and length.
VSAM	SQL	Joined fields must have a common datatype and length.
QSAM	SQL	Joined fields must have a common datatype and length.

Note:

- A QSAM file is a sequential file. One common example is a FOCUS HOLD file.
- When joining to a VSAM, QSAM, or IMS data structure, the field in the host file can be shorter in length than the field in the cross-referenced file, but performance is adversely affected. This is the only exception to the “same datatype, same length” rule when joining non-relational files.
- For efficiency, create RDBMS indexes on host and cross-referenced fields. Check with the database administrator or query the index catalog table to see which columns are indexed.
- Text fields (TX) may not participate as host or cross-referenced fields.
- In report requests, specify the name of the host structure in the TABLE FILE statement, not the join name specified with the AS phrase.
- You can construct a dynamic join based on more than one field. See *Multi-field Dynamic Join* on page 8-14, for multi-field joins.
- For information about how heterogeneous joins affect Interface optimization, see *Optimization of Joins Between Heterogeneous File Types* in Chapter 7, *The Interface Optimizer*.

Example Using the Dynamic JOIN Command

The following examples illustrate the use of the FOCUS dynamic JOIN command. Each example specifies a FOCUS non-unique join and presents an equivalent SQL request for comparison purposes. Both forms of the request, FOCUS and SQL, return the same result.

Note:

- The EMPINFO and COURSE tables are described in Appendix C, *File Descriptions and Tables*.
- You can reproduce the SQL requests with the FSTRACE4 trace. See Appendix D, *Tracing Interface Processing*, for trace syntax.

The first example prints each employee's full name and courses taken. Since the employee may have taken more than one course, the example specifies the FOCUS non-unique join.

Employee information is stored in the EMPINFO table and is represented by the fields LAST_NAME and FIRST_NAME. Course information is stored in the COURSE table and is represented by the CNAME field.

```
 >  > tso alloc f(fstrace4) da(*) lrecl(80) recfm(f)
 >  > join emp_id in empinfo tag file1 to all who in course tag file2 as j1
 >  > table file empinfo
 > print cname
 > by last_name by first_name
 > end
    SELECT T1.EID,T1.LN,T1.FN,T2.COURSE_NAME FROM
   "USER1"."EMPINFO" T1,"USER1"."COURSE" T2 WHERE (T2.EMP_NO =
   T1.EID) ORDER BY T1.LN,T1.FN FOR FETCH ONLY;
```

The SQL request references both tables in the FROM clause and as a condition (or join predicate) of the WHERE clause. The FOCUS BY phrases translate to the ORDER BY phrase. The PRINT verb translates as SELECT.

The next example illustrates a screening test that lists the employees who have taken either the Advanced or Internals course.

```
join who in course tag file1 to all emp_id in empinfo tag file2 as j2
 >  > table file course
 > print cname
 > by last_name by first_name
 > where cname is 'advanced' or 'internals'
 > end
    SELECT T1.COURSE_NAME,T1.EMP_NO,T2.LN,T2.FN FROM
   "USER1"."COURSE" T1,"USER1"."EMPINFO" T2 WHERE (T2.EID =
   T1.EMP_NO) AND (T1.COURSE_NAME IN('ADVANCED','INTERNALS'))
   ORDER BY T2.LN,T2.FN FOR FETCH ONLY;
```

The SQL request references both tables in the FROM clause and in a join predicate in the WHERE clause (along with additional screening conditions). The FOCUS BY phrases translate to the ORDER BY phrase.

Multi-field Dynamic Join

You can construct a dynamic join based on multiple fields from the host file and cross-referenced files. Separate the participating fieldnames with the keyword AND.

The syntax for the multi-field JOIN is

```
JOIN fielda1 AND fielda2 IN hostfile [TAG tag1] TO [ALL]
fieldb1 AND fieldb2 IN crfile [TAG tag2] AS joinname
[END]
```

where:

`fielda1`
: Is a field in the host file that shares values and format with fieldb1 in the cross-referenced file.

`AND fielda2`
: Is a field in the host file that shares values and format with fieldb2 in the cross-referenced file.

`hostfile`
: Is the name of the host file. Use this name in subsequent TABLE requests on the joined structure. The host file can be any type of database or table. See *Single-field Dynamic Join* on page 8-10 for possible join combinations.

`tag1`
: Is the tag name for the host file, a one- to eight-character table name qualifier for field and alias names in the host file. The tag name for the host must be the same in all the JOIN commands for a join structure. The tag name may not be the same as any table name in the structure

`ALL`
: Indicates that the host and cross-referenced files participate in a one-to-many or many-to-many relationship. That is, for any instance of the join fields in the host file (fielda1, fielda2), there may be more than one corresponding instance of the join fields in the cross-referenced file (fieldb1, fieldb2). In FOCUS terminology, this is known as a non-unique or multiple join.

  Note: The use of the ALL parameter does not disable optimization. Do not confuse this ALL parameter with the SET ALL command discussed in *The SET ALL Command* on page 8-18.

  Omitting the ALL parameter indicates a unique join. Omit the ALL parameter only when each row in the host file has, at most, one corresponding row in the cross-referenced file. The unique join is a FOCUS concept; the RDBMS makes no distinction between unique and non-unique situations when it processes a join.

`fieldb1`
: Is a field in the cross-referenced file whose values and format can match that of fielda1.

`AND fieldb2`
: Is a field in the cross-referenced file whose values and format can match that of fielda2.

`crfile`
: Is the name of the cross-referenced file.

`tag2`
: Is the tag name for the cross-referenced file, a one- to eight-character tablename qualifier for field and alias names in the cross-referenced file. The tag name may not be the same as any tablename in the structure. In a recursive join structure, if no tagname is provided, FOCUS prefixes all fieldnames and aliases with the first four characters of the joinname.

joinname

Assigns an internal name to the join structure, up to eight characters in length. It also provides a unique prefix for fieldnames participating in a recursive join.

You can clear the join name when you no longer need this join (see *JOIN CLEAR* on page 8-31). Do not specify joinname as the filename in subsequent TABLE FILE requests; use hostfile.

END

Required when the JOIN statement is longer than one line; terminates the statement.

For a multi-field join, the joined fields from each table must be equal in number, datatype, and field length. The full set of common fields must reside in both the host and cross-referenced tables.

You can join a maximum of 16 fields from a host file to 16 fields in a cross-referenced file. Each multi-field JOIN command counts as one join toward the FOCUS maximum of 16 concurrent joins.

Note: For a join to be optimized, it cannot involve more than 15 tables or views, the RDBMS limit for an SQL statement.

Example Using a Multi-field Dynamic Join

For example, two RDBMS tables, EMPINFO and COURSE1, have first and last name fields. In the EMPINFO Master File, LAST_NAME and FIRST_NAME have field formats A15 and A10:

```
FILENAME=EMPINFO      ,SUFFIX=SQLDS,$

SEGNAME=EMPINFO       ,SEGTYPE=S0,$
 FIELD=EMP_ID  ,ALIAS=EID     ,USAGE=A9      ,ACTUAL=A9,$
 FIELD=LAST_NAME      ,ALIAS=LN      ,USAGE=A15     ,ACTUAL=A15,$
 FIELD=FIRST_NAME     ,ALIAS=FN      ,USAGE=A10     ,ACTUAL=A10,$
 FIELD=HIRE_DATE      ,ALIAS=HDT     ,USAGE=YMD     ,ACTUAL=DATE,$
 FIELD=DEPARTMENT     ,ALIAS=DPT     ,USAGE=A10     ,ACTUAL=A10,
    MISSING=ON,$
 FIELD=CURRENT_SALARY ,ALIAS=CSAL    ,USAGE=P9.2    ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE   ,ALIAS=CJC     ,USAGE=A3      ,ACTUAL=A3,$
 FIELD=ED_HRS  ,ALIAS=OJT     ,USAGE=F6.2    ,ACTUAL=F4,    MISSING=ON,$
 FIELD=BONUS_PLAN     ,ALIAS=BONUS_PLAN     ,USAGE=I4      ,ACTUAL=I4,$
```

In the COURSE1 Master File, LAST_NAME and FIRST_NAME have the same field formats, A15 and A10:

```
FILE=COURSE1     ,SUFFIX=SQLDS, $
SEGNAME=SEG01    ,SEGTYPE=S0, $
FIELDNAME   =FOCLIST          ,FOCLIST     ,I5       ,I4       ,$
FIELDNAME   =CNAME            ,COURSE_NAME ,A15      ,A15      ,$
FIELDNAME   =LAST_NAME        ,LN          ,A15      ,A15      ,$
FIELDNAME   =FIRST_NAME       ,FN          ,A10      ,A10      ,$
FIELDNAME   =GRADE            ,GRADE       ,A1       ,A1       ,
      MISSING=ON, $
FIELDNAME   =YR_TAKEN         ,YR_TAKEN    ,A2       ,A2       ,$
FIELDNAME   =QTR              ,QUARTER     ,A1       ,A1       ,$
```

Note: Appendix C, *File Descriptions and Tables*, does not include the COURSE1 Master File. To reproduce the COURSE1 Master File, extract the fields from the ECOURSE Master File and use the HOLD FORMAT SQL option explained in *Creating Extract Files on the RDBMS* on page 8-3.

To create the multi-field join, list both fields for each table with the keyword AND:

```
JOIN LN AND FN IN EMPINFO TAG FILE1
TO ALL LAST_NAME AND FIRST_NAME IN COURSE1 TAG FILE2 AS J1
END
```

When a report request references the multi-field dynamic join, the Interface generates an SQL SELECT statement to satisfy the request:

```
join ln and fn in empinfo tag file1
 > to all last_name and first_name in course1 tag file2 as j1
 > end
 >  > table file empinfo
 > print ln fn cname
 > end
    SELECT T1.LN,T1.FN,T2.COURSE_NAME FROM "USER1"."EMPINFO" T1,
   "USER1"."COURSE1" T2 WHERE (T2.LN = T1.LN) AND (T2.FN = T1.FN) FOR FETCH
   ONLY;
```

The SQL SELECT statement implements two joins as conditions (or predicates) of the WHERE clause.

Missing Rows of Data in Cross-referenced Tables

This topic describes factors that affect report results when a host row has no corresponding cross-referenced row. The discussion applies to both dynamic and embedded joins.

Normally, when a row from the host table or view is retrieved, a corresponding row from the cross-referenced table can also be retrieved. If a host row lacks a corresponding cross-referenced row, the retrieval path is called a **short path**. When there are short paths, the processing of the host row and the report results depend on:

- The FOCUS SET ALL command (or the use of the ALL. prefix).
- Whether Interface optimization is enabled or disabled. (See Chapter 7, *The Interface Optimizer*, for information about the SET OPTIMIZATION command.)

  Note: Even if you set OPTIMIZATION ON, the Interface may disable it. Use FSTRACE3 to determine whether optimization is enabled for a particular request (Appendix D, *Tracing Interface Processing*, describes trace facilities).
- The type of join: non-unique or unique.

The SET ALL Command

You can include short paths in a report with the FOCUS SET ALL command

```
SET ALL = {OFF|ON}
```

where:

`OFF`
: Is the default. In a join, omits host rows from the report if they lack a corresponding cross-referenced row.

`ON`
: Includes all host rows in the report. (This is known as a left outer join.)

Note: The Interface does not support SET ALL = PASS.

Controlling Outer Join Optimization

With the SET SQLJOIN OUTER command you can control when the Interface optimizes outer joins without affecting the optimization of other operations. This parameter provides backward compatibility with prior releases of the Interface and enables you to fine-tune your applications.

When join optimization is in effect, the Interface generates one SQL SELECT statement that includes every table involved in the join. The RDBMS can then process the join. When join optimization is disabled, the Interface generates a separate SQL SELECT statement for each table, and FOCUS processes the join.

You can use the SQLJOIN OUTER setting to disable outer join optimization while leaving other optimization enhancements in effect.

Syntax How to Control Outer Join Optimization

Issue the following command

```
SQL target_db SET SQLJOIN OUTER {ON|OFF}
```

where:

target_db

Indicates the target RDBMS. Valid values are DB2 or SQLDS. Omit if you issued the SET SQLENGINE command.

`ON`

Enables outer join optimization. ON is the default value.

`OFF`

Disables outer join optimization.

Note:

- The SQLJOIN OUTER setting is available only when optimization is enabled (that is, OPTIMIZATION is not set to OFF).
- The SQLJOIN OUTER setting is ignored when SET ALL = OFF.

Effects of Combinations of Settings on Outer Join Optimization

The following table describes how different combinations of OPTIMIZATION and SQLJOIN OUTER settings affect Interface behavior. It assumes that SET ALL = ON:

Settings		**Results**	
OPTIMIZATION	**SQLJOIN OUTER**	**Outer Join Optimized?**	**Other Optimization Features**
ON	ON	Yes	Enabled
ON	OFF	No	Enabled
OFF	N/A	No	Disabled
SQL	ON	Yes, in all possible cases	Enabled
SQL	OFF	No	Enabled
FOCUS	ON	Yes if results are equivalent to FOCUS managed request	Enabled
FOCUS	OFF	No	Enabled

Reference SQLJOIN OUTER Messages

If SQLJOIN OUTER is set to OFF, the following message displays when you issue the SQL ? query command:

```
(FOC1420) OPTIMIZATION OF ALL=ON AS LEFT JOIN -  : OFF
```

Missing Rows in Unique Descendants

In a unique join, the engine that handles the join controls the output:

- IF FOCUS handles the join, it treats a unique cross-referenced table as a logical extension of the host or parent table. Since FOCUS never considers a unique cross-referenced row to be missing, it displays short paths regardless of whether SET ALL is ON or OFF. FOCUS handles the join in the following two cases:
    - OPTIMIZATION is disabled.
    - OPTIMIZATION is enabled with SET ALL = ON and SQLJOIN OUTER OFF.

      In this case the SQLJOIN OUTER OFF setting requires that FOCUS process the left outer join even though other optimization features remain in effect.

  Note: When FOCUS handles join processing, if you describe a join as unique when the cross-referenced table actually contains more than one matching record for a row in the host table, FOCUS displays only the first matching cross-referenced row on the report. Do not specify a unique join to FOCUS when the data structure is non-unique.

- If the RDBMS handles the join, OPTIMIZATION must be enabled. The SET ALL command determines the output:
    - If SET ALL = OFF, the RDBMS performs an inner join; it omits short paths from the report and includes multiple matching cross-referenced rows.
    - If SET ALL = ON the SQLJOIN OUTER setting determines whether FOCUS or the RDBMS process the join. When SQLJOIN OUTER is ON, the RDBMS processes the left outer join; it includes all host rows in the report and includes multiple matching cross-referenced rows.

      Since the RDBMS has no concept of a unique join, its behavior is identical whether the join is unique or non-unique. See *Missing Rows in Non-unique Descendants* on page 8-23 for a complete discussion.

The following topics discuss each of these settings.

FOCUS Unique Join Processing

FOCUS handles the join when either of the following conditions is true:

- OPTIMIZATION disabled.
- OPTIMIZATION enabled, SET ALL = ON, and SQLJOIN OUTER OFF.

FOCUS substitutes default values for any missing cross-referenced data (because it does not consider them to be missing in a unique join). FOCUS recognizes that you have defined the join to be unique and:

- It displays short paths with blanks or zeros in missing columns.
- It includes only the first instance found of any multiple matching cross-referenced rows.

Chapter 7, *The Interface Optimizer,* discusses factors that determine whether optimization is disabled. To find out if and why the Interface disabled OPTIMIZATION for a report request, use FSTRACE3 (see Appendix D, *Tracing Interface Processing*).

In the following example, a unique join connects the EMPINFO table, containing employee information, to the FUNDTRAN table, containing direct deposit account information for the employees. (Appendix C, *File Descriptions and Tables*, provides the Master and Access Files.)

```
join emp_id in empinfo tag file1 to who in fundtran tag file2 as j1
>
sql db2 set optimization off
>
table file empinfo
print bank_name bank_acct
by department by emp_id
end

NUMBER OF RECORDS IN TABLE=       14  LINES=     14
```

Since the join is unique and the SET OPTIMIZATION OFF command disables optimization, FOCUS displays one row per employee on the report. If there is no data for the cross-referenced table (FUNDTRAN), FOCUS appropriately substitutes zeros or blanks for its fields.

Two new employees, John Royce (333121200, no department) and Donna Lee (455670000, MIS) have no bank accounts. Six other employees also lack bank accounts:

```
PAGE     1

DEPARTMENT      EMP_ID        BANK_NAME           BANK_ACCT
----------      ------        ---------           ---------
.               333121200                                 0
MIS             112847612                                 0
                117593129     STATE                40950036
                219984371                                 0
                326179357     ASSOCIATED          122850108
                455670000                                 0
                543729165                                 0
                818692173     BANK ASSOCIATION    163800144
PRODUCTION      071382660                                 0
                119265415                                 0
                119329144     BEST BANK              160633
                123764317     ASSOCIATED          819000702
                126724188                                 0
                451123478     ASSOCIATED          136500120
```

Note: Employee 333121200 has not yet been assigned a department; its null value is indicated by the NODATA symbol. However, BANK_NAME and BANK_ACCT are not considered missing because the join is unique.

RDBMS Unique Join Processing

The RDBMS processes the join under the following conditions:

- OPTIMIZATION enabled with SET ALL = OFF.

  Since the RDBMS does not recognize the concept of a unique join, it performs an inner join just as it does if the join is non-unique. See *Missing Rows in Non-unique Descendants* on page 8-23 for a complete discussion.

- OPTIMIZATION enabled with SET ALL = ON and SQLJOIN OUTER ON.

  If SQLJOIN OUTER is ON, the RDBMS performs a left outer join just as it does if the join is non-unique. See *Missing Rows in Non-unique Descendants* on page 8-23 for a complete discussion.

  (If SQLJOIN OUTER is OFF, FOCUS processes the join. It substitutes default values for short path columns and displays only one matching record as described in *FOCUS Unique Join Processing* on page 8-21.)

Chapter 7, *The Interface Optimizer,* discusses factors that determine whether optimization is enabled.

Missing Rows in Non-unique Descendants

In a non-unique join (JOIN…TO ALL) with missing cross-referenced rows, report results depend entirely on the SET ALL command because:

- If SET ALL=OFF, the RDBMS and FOCUS both perform an inner join. Therefore, the OPTIMIZATION setting has no effect on the report results. Short paths are omitted from the report; multiple matching rows are included.
- If SET ALL=ON, the RDBMS and FOCUS both perform a left outer join. Therefore, the OPTIMIZATION setting has no effect on the report results (although if optimization is enabled, you can control which engine processes the outer join with the SET SQLJOIN OUTER command described in *How to Control Outer Join Optimization* on page 8-19). Short paths display in the report with missing columns represented by the NODATA symbol (.); multiple matching rows are included.

SET ALL=OFF With a Non-unique Join

With SET ALL=OFF, optimization may be enabled or disabled. In both cases, however, the report results are the same: an inner join. Host rows that lack corresponding cross-referenced rows are not included in the report; multiple matching rows are included (even if there is only one).

For each of the following examples, the same non-unique join is in effect. It connects the EMPINFO table, containing employee information, to the DEDUCT table, containing salary deduction information.

The examples also execute the same report request. OPTIMIZATION is set ON and only the SET ALL command changes.

Two employees, John Royce (333121200, no department) and Donna Lee (455670000, MIS) have not been paid yet; therefore, they have no deductions.

When SET ALL is OFF, a host row that lacks a corresponding cross-referenced row is rejected (regardless of whether FOCUS or the RDBMS processes the join):

```
join clear j1
>
join emp_id in empinfo tag file1 to all dedeid in deduct tag file2 as j2
>
set all=off
>
table file empinfo
print ded_code ded_amt
by department by emp_id by deddate
end

NUMBER OF RECORDS IN TABLE=      448  LINES=    448
```

The report displays multiple deduction records in the cross-referenced table for each of 12 long-time employees in the host table. John Royce (333121200, no department) and Donna Lee (455670000, MIS department) are not listed in the report, because their corresponding cross-referenced rows do not exist in the DEDUCT table:

```
PAGE     1

DEPARTMENT  EMP_ID     DEDDATE   DED_CODE  DED_AMT
----------  ------     -------   --------  -------
MIS         112847612  82/01/29  CITY        $1.43
                                 SAVE       $54.60
                                 STAT       $20.02
                                 LIFE       $13.65
                                 HLTH       $22.75
                                 FICA      $100.10
                                 FED       $121.55
                       82/02/26  STAT       $20.02
                                 SAVE       $54.60
                                 LIFE       $13.65
                          .
                          .
                          .
```

SET ALL=ON With a Non-unique Join

With a non-unique join, SET ALL=ON produces a left outer join regardless of whether FOCUS or the RDBMS handles the join:

- If optimization is enabled and SQLJOIN OUTER is ON, the Interface passes a left outer join to the RDBMS.
- If optimization is disabled or optimization is enabled but SQLJOIN OUTER is OFF, FOCUS handles the join. Honoring the SET ALL command, FOCUS includes all host rows; since the join is non-unique, it also includes multiple matching rows.

In both cases the report results are the same.

If there are no cross-referenced rows for a host row, the cross-referenced columns display the NODATA value.

The following example executes the same report request as in *SET ALL=OFF With a Non-unique Join* on page 8-23, except that SET ALL is ON:

```
set all = on
 >  > table file empinfo
 > print ded_code ded_amt
 > by department by emp_id by deddate
 > end
  NUMBER OF RECORDS IN TABLE=      450  LINES=    450
```

John Royce (333121200) is now on Page 1 and Donna Lee (455670000) on Page 8. The department column for John Royce (333121200) displays the NODATA value, since the field has MISSING=ON:

```
PAGE     1
   DEPARTMENT    EMP_ID      DEDDATE      DED_CODE     DED_AMT
   ----------    ------      -------      --------     -------
   .             333121200   .            .                  .
   MIS           112847612   82/01/29     CITY           $1.43
                                          FED          $121.55
                                          FICA         $100.10
                                          HLTH          $22.75
                                          LIFE          $13.65
                             .
                             .
                             .
```

Donna Lee (455670000) works for MIS:

```
PAGE     8
   DEPARTMENT    EMP_ID      DEDDATE     DED_CODE    DED_AMT
   ----------    ------      -------     --------    -------
   MIS           326179357   82/07/30    STAT         $88.93
                             82/08/31    CITY          $6.35
                                         FED         $539.96
                                         FICA        $444.67
                                         HLTH         $45.37
                                         LIFE         $27.22
                                         SAVE        $108.90
                                         STAT         $88.93
                 455670000   .           .                 .
                 543729165   82/04/30    CITY           $.72
                             .
                             .
                             .
```

Note: The report that results from passing a request to the RDBMS with SET ALL=ON may be sorted in a slightly different order from the report produced if FOCUS processes the join; however, both results are equally correct.

SET ALL=ON With Screening Conditions

When SET ALL=ON, screening conditions affect report results for a non-unique join. If a screening test specifies a field from the cross-referenced structure, host rows whose cross-referenced rows were screened out are not represented, regardless of the SET ALL command.

The following example includes a WHERE test to screen for health deduction records:

```
join emp_id in empinfo tag file1 to all dedeid in deduct tag file2 as join1
set all=on

table file empinfo
> sum ded_amt
> by department by ln by fn by ded_code
> where ded_code eq 'hlth'
> end
NUMBER OF RECORDS IN TABLE=      56  LINES=    10
```

Royce and Lee are omitted as the result of the WHERE test, because they have no deduction code records:

```
   DEPARTMENT     LAST_NAME    FIRST_NAME     DED_CODE    DED_AMT
   ----------     ---------    ----------     --------    -------
   MIS            BLACKWOOD    ROSEMARIE      HLTH        $226.87
                  CROSS        BARBARA        HLTH        $330.21
                  JONES        DIANE          HLTH        $121.16
                  MCCOY        JOHN           HLTH         $16.50
                  SMITH        MARY           HLTH        $181.99
   PRODUCTION     BANNING      JOHN           HLTH         $41.25
                  IRVING       JOAN           HLTH        $427.35
                  MCKNIGHT     ROGER          HLTH        $122.43
                  ROMANS       ANTHONY        HLTH        $105.60
                  STEVENS      ALFRED         HLTH         $69.44
```

Selective ALL. Prefix

Even if SET ALL=OFF, you can apply the effect of the ON setting to specific tables. To do this, add the ALL. prefix to one of the *host* fields in the request. The ALL. prefix causes FOCUS to process host rows even if they have missing cross-referenced rows. Like SET ALL=ON, the report results depend on whether screening tests exist for the cross-referenced fields.

For example, when the ALL. prefix is applied to the field DEPARTMENT, the report results are the same as for SET ALL=ON. (The report is displayed in *SET ALL=ON With a Non-unique Join* on page 8-24).

```
>
set all=off
>
table file empinfo
print ded_code ded_amt
by all.department by emp_id by deddate
end

   NUMBER OF RECORDS IN TABLE=      450  LINES=    450
```

Note: The ALL. prefix is only effective when the SET ALL value is OFF; SET ALL=ON overrides the ALL. prefix.

Summary Chart

This summary chart lists possible report results for dynamic joins and multi-table Master Files (embedded joins):

	RDBMS Behavior (Optimization Enabled)		**FOCUS Behavior (Optimization Disabled)**	
	SET ALL =		**SET ALL =**	
Join Type	**ON (SQLJOIN OUTER ON)**	**OFF**	**ON**	**OFF**
NON-UNIQUE Dynamic (`JOIN...TO ALL`) or Embedded (`SEGTYPE=S0 or KL`)	**Outer Join**	**Inner Join**	**Outer join** **(Also applies if Optimization Enabled and SQLJOIN OUTER OFF)**	**Inner join**
Short paths	Appear with NODATA (.) value for fields of all missing segments	Do not appear	Appear with NODATA (.) value for fields of all missing segments	Do not appear
More than one matching row	All matching rows appear	All matching rows appear	All matching rows appear	All matching rows appear
UNIQUE Dynamic (`JOIN...TO`) or Embedded (`SEGTYPE=U or KLU`)	**Outer join**	**Inner join**	**FOCUS Unique** **(Also applies if Optimization Enabled and SQLJOIN OUTER OFF)**	**FOCUS Unique**
Short paths	Appear with NODATA (.) value for fields of all missing segments	Do not appear	Appear with blank or zero for missing values	Appear with blank or zero for missing values
More than one matching row	All matching rows appear	All matching rows appear	Only first matching row appears	Only first matching row appears

Note:

- For non-unique dynamic and embedded joins, use SET ALL=ON to display the short paths.
- For unique dynamic and embedded joins, use either OPTIMIZATION OFF or SET ALL=ON to display the short paths.
- For unique dynamic and embedded joins, when OPTIMIZATION is OFF or SET ALL = ON with SQLJOIN OUTER OFF, FOCUS produces an outer join unless there is more than one matching row. If there is more than one matching row, only the first matching row appears.

JOIN Utilities

You can check and control the status of join structures with three commands, CHECK FILE, ? JOIN query, and JOIN CLEAR.

CHECK FILE

The CHECK FILE command produces a diagram of host and cross-referenced relationships and retrieval paths.

To display the structure, at the FOCUS command level type

```
CHECK FILE name PICT[URE] [RETRIEVE]
```

where:

name

Is the name of the host structure (for a dynamic join) or of a multi-table Master File.

RETRIEVE

Is optional; modifies the diagram to show retrieval paths, especially how a unique table is treated as an extension of its host. See the *FOCUS for IBM Mainframe User’s Manual* for more information.

On the diagram, labels for non-unique (KM) or unique (KU) next to the cross-referenced structure indicate whether the dynamic JOIN is non-unique or unique. Only the first four fieldnames of each structure display.

The letter K next to a field indicates that it represents the foreign key column or the first column of a composite foreign key. If the cross-referenced structure has descendants, the descendants are labeled as either KL (keyed through linkage) or KLU (keyed through linkage unique), depending on how their relationship was declared to FOCUS.

For example:

```
join emp_id in empinfo tag file1 to all who in course tag file2 as j1
 >
 check file empinfo pict

  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   2  ( REAL=    1  VIRTUAL=   1 )
  NUMBER OF FIELDS=    13  INDEXES=   0  FILES=     2
  TOTAL LENGTH OF ALL FIELDS=   95

 SECTION 01
 STRUCTURE OF GNTINT   FILE EMPINFO ON 06/19/90 AT 12.17.40

          EMPINFO
  01      S0
 **************
 *EMP_ID      **
 *LAST_NAME   **
 *FIRST_NAME  **
 *HIRE_DATE   **
 *            **
 ***************
  **************
        I
        I
        I
        I COURSE
  02    I KM
 ..............
 :CNAME       ::
 :WHO         ::K
 :GRADE       ::
 :YR_TAKEN    ::
 :            ::
 :............::
  .............:
  JOINED  COURSE
```

Note: “JOINED COURSE” indicates that this relationship was created by a dynamic JOIN command rather than a multi-segment Master File.

? JOIN

For all dynamic joins that are in effect, the ? JOIN query command identifies the fields representing the primary/foreign key field pair, the host structure, the cross-referenced structure, any specified join name, and the type of join (non-unique or unique, as indicated by the presence or absence of the ALL keyword).

To list all active dynamic join structures, enter:

```
? JOIN
```

For example:

```
? join
  JOINS CURRENTLY ACTIVE

 HOST                          CROSSREFERENCE
 FIELD        FILE     TAG      FIELD        FILE     TAG      AS         ALL
 -----        ----     ---      -----        ----     ---      --         ---
 LN           EMPINFO  FILE1    LAST_NAME    COURSE1  FILE2    J1         Y
```

Note: The ? JOIN query command displays only the first join pair for multi-field joins.

JOIN CLEAR

A maximum of 16 dynamic joins can be active in any FOCUS session. The JOIN CLEAR command can release either all or specific existing joins. The syntax is

```
JOIN CLEAR {* | joinname}
```

where:

`*`

Clears all existing joins.

`joinname`

Is a specific join to be cleared.

Search Limits

The FOCUS READLIMIT and RECORDLIMIT features set a maximum for the number of rows the Interface fetches from the answer set returned by the RDBMS. They do not restrict the RDBMS in its construction of the answer set. However, if the RDBMS does not have to sort the answer set, using these features may result in a significant reduction in response time.

The READLIMIT and RECORDLIMIT features are effective in reducing RDBMS-to-FOCUS communication traffic, the volume of formatted report data, and terminal and disk I/Os.

Search limit tests are helpful when:

- Testing a new Master File. Reporting requires only a few rows to indicate if the description is accurate.
- Using the FSTRACE facility in problem resolution.
- Testing the format of a new report.

In all three cases, a few rows are sufficient to verify results.

To restrict the maximum number of SQL fetch commands performed for RDBMS-returned data, add the following phrase to the TABLE request

```
IF READLIMIT IS n
```

where:

n

Is the number of records to be included in the TABLE request.

READLIMIT appends an OPTIMIZE FOR n ROWS clause to the SQL generated by the Interface. This can improve DB2 performance if you know the size of the desired answer set in advance. For more information, consult the IBM *DB2 SQL Reference* Manual.

To restrict the maximum number of SQL fetch commands performed for RDBMS-returned data without appending the OPTIMIZE FOR n ROWS clause, use the following phrase in the TABLE request instead

```
IF RECORDLIMIT IS n
```

where:

n

Is the number of records to be included in the TABLE request.

The FOCUS database administrator may place the READLIMIT or RECORDLIMIT test in a Master File to limit the fetching of rows for all users. If the Master File and the report request both contain such a test, the Interface uses the lower of the two values.

Note: In some instances, the RDBMS performs a substantial volume of work before returning any data to FOCUS. To limit the amount of work performed by the RDBMS, add another IF or WHERE test to one or more of the tables in the request.

Multiple Retrieval Paths

This section discusses retrieval performance for multi-table Master Files and dynamic joins of three or more tables. It also explains the role of unique cross-referenced tables.

Master Files and JOIN commands define data retrieval paths. The Interface queries only those RDBMS tables necessary for the report. These include tables containing fields specified in the request plus any connecting tables needed to construct a retrieval plan (subtree). The retrieval sequence of a subtree is top to bottom, left to right.

Note: A FOCUS TABLE request that references a dynamically joined structure generates SQL join predicates for all segments in the subtree that starts from the root segment. Multi-table Master Files do not necessarily generate these predicates; in a multi-table structure, the subtree effectively begins with the highest referenced segment. This difference may cause identical TABLE requests to produce different reports when run against a dynamic join structure and a multi-table Master File that represent the same tree structure. See Chapter 8, *Advanced Reporting Techniques*, for a discussion of dynamic joins.

FOCUS treats unique tables as extensions of their hosts. In cases where the host has both unique and non-unique cross-referenced tables, FOCUS always retrieves the unique cross-referenced tables first.

To display retrieval paths, use the CHECK FILE command with the RETRIEVE option. (See *CHECK FILE* on page 8-29).

Multiple Retrieval Paths With Sort Phrases and Screening Tests

Fields specified in sort phrases (BY and ACROSS) and screening tests (IF and WHERE) must lie on the same retrieval path as the other requested fields. That is, the table containing the BY field or column being screened must precede or follow other tables referenced in the request. A field that is not on the same path generates FOCUS error message FOC029.

9 Direct SQL Passthru

Topics:

- Invoking Direct SQL Passthru
- Issuing Commands and Requests
- Parameterized SQL Passthru

With the Direct SQL Passthru facility, you can issue any native SQL command directly to the RDBMS from FOCUS. This facility, included with the Interface, provides Interface support for both native SQL and Interface environmental commands. You must know native SQL to use this feature for issuing SQL statements, but not for the Interface SET commands.

Issuing requests through the Direct SQL Passthru facility eliminates the need for FOCUS Master and Access Files but retains all FOCUS reporting capabilities.

Note: Direct SQL Passthru is not available within MODIFY.

Direct SQL Passthru provides the following advantages:

- Support for native SQL commands, including SQL SELECT statements.

  With Direct SQL Passthru, FOCUS provides a storage area for the RDBMS-supplied data (answer sets) that result from SELECT statements. Therefore, you can issue SELECT statements.

- Support for all SQL SELECT options.

  You can issue any SELECT syntax supported by the RDBMS, regardless of whether an equivalent FOCUS operation exists. Neither the Interface nor the Direct SQL Passthru facility translates SQL to FOCUS in order to process a request.

  For example, no FOCUS syntax exists that would cause the Interface to generate one SELECT with a UNION or with a subquery. The Direct SQL Passthru facility supports both operations.

- Support for parameter markers in Direct SQL Passthru commands, so you can execute them repeatedly with varying input values.

- An alternative to the FOCUS SQL Translator.

  Some applications use the FOCUS SQL Translator to access the RDBMS via a FOCUS Interface. The Translator translates SQL to FOCUS, and then the Interface uses this FOCUS code to generate SQL that it passes to the RDBMS. The Direct SQL Passthru facility bypasses internal translation and offers a more direct means of accessing the RDBMS.

The Translator supports one SQL dialect, ANSI standard Level 2 SQL. You can use the Translator to produce reports from any data source that is described to FOCUS (for example, IMS, a FOCUS database, or the DB2 RDBMS). For more information about the FOCUS SQL Translator and access to FOCUS databases, see the *FOCUS for IBM Mainframe User's Manual.*

Note: If your site installed the Interface with the IM parameter set to 0, certain SQL commands will be disabled. Issuing these commands will produce error message FOC1638. You can issue SQL SELECT commands regardless of the IM setting; they are never disabled. Refer to the *FOCUS for IBM Mainframe DB2 and SQL/DS Read/Write Interface Installation Guide* for more information.

Invoking Direct SQL Passthru

To invoke the Direct SQL Passthru facility, you must establish the target RDBMS in either of the following two ways:

- Issue the Interface SET SQLENGINE command to establish the target RDBMS for the duration of the FOCUS session or until you issue another SET SQLENGINE command. The Interface automatically passes subsequent SQL commands to the specified RDBMS.
- Specify the target RDBMS in your request. (See *Issuing Commands and Requests* on page 9-2 for information about issuing commands and requests.)

The syntax for the SET SQLENGINE command is

```
SET SQLENGINE = target
```

where:

target
Indicates the target RDBMS. Valid values are as follows:

<u>OFF</u>	Indicates that the FOCUS SQL Translator will process the request. OFF is the default setting.
DB2	Indicates the DB2 RDBMS.
SQLDS	Indicates the SQL/DS RDBMS.

You can change the SET SQLENGINE setting at any point in your FOCUS session.

Issuing Commands and Requests

If you do not issue the SET SQLENGINE command, you must specify the target RDBMS in any SQL command you want to pass directly to the RDBMS.

The following is a request syntax summary for native SQL commands, including SELECT statements, and for Interface environmental commands; subsequent sections provide examples

```
SQL [target_db]
command [;]
[TABLE FILE SQLOUT]
[options]
END
```

where:

target_db
: Indicates the target RDBMS. Acceptable values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

command
: Is one SQL command, or one or more Interface SET commands.

;
: For SQL SELECT requests only, the semicolon is required if you specify additional FOCUS report options.

`TABLE FILE SQLOUT`
: For SQL SELECT requests only, allows you to specify additional FOCUS report options or subcommands. To create a Master File you can use throughout the FOCUS session, see *Creating FOCUS Views With Direct SQL Passthru* on page 9-12.

options
: For SQL SELECT requests only, are report formatting options or operations.

`END`
: Terminates the request. Is optional for Interface SET commands, the SQL commands COMMIT WORK and ROLLBACK WORK, the DB2 CONNECT command, and the Interface parameterized Passthru commands BEGIN SESSION, END SESSION, and PURGE (see *Parameterized SQL Command Summary* on page 9-15). Required for all other commands.

Including a target RDBMS in your command overrides the SET SQLENGINE command. For example, you can specify your RDBMS and override an existing OFF setting. If you do not specify a target RDBMS (either in your command or with the SET SQLENGINE command), the SQL keyword invokes the Translator.

Note:

- The OFF setting is valid only for the SET SQLENGINE command.
- You cannot issue an SQL command together with Interface SET commands in one request.
- Do not prefix Interface SET commands with environmental qualifiers (TSO, MVS, or CMS) when you use the Direct SQL Passthru facility.

Displaying the Effects of UPDATE and DELETE Commands

You can use the SET PASSRECS command to display the number of rows affected by a successfully executed Direct SQL Passthru UPDATE or DELETE command. The syntax is

```
SQL [target_db] SET PASSRECS {OFF|ON}
```

where:

target_db

Indicates the target RDBMS. Valid values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

OFF

Is the default. As in previous releases, the Interface provides no information as to the number of records affected by a successfully executed Direct SQL Passthru UPDATE or DELETE command.

ON

Provides the following FOCUS message after the successful execution of a Direct SQL Passthru UPDATE or DELETE command:

```
(FOC1364) ROWS AFFECTED BY PASSTHRU COMMAND: #/operation
```

For example, a DELETE command that executes successfully and affects 20 rows generates the following message:

```
(FOC1364) ROWS AFFECTED BY PASSTHRU COMMAND: 20/DELETE
```

In addition to this message, the Interface updates the &RECORDS FOCUS system variable with the number of rows affected. You can access this variable via Dialogue Manager, and display it with the ? STAT query.

Note:

- Although you can use native SQL syntax (see Appendix B, *Native SQL*) to issue the SET PASSRECS command, if you use it to issue subsequent UPDATE or DELETE commands, the Interface will not issue the FOC1364 message or update the &RECORDS system variable. You must use Direct SQL Passthru syntax to issue UPDATE or DELETE commands in order to invoke the SET PASSRECS command.
- Since, by definition, the successful execution of an INSERT command always affects one record, INSERT does not generate the FOC1364 message.
- The FOC1364 message is for informational purposes only and does not affect the &FOCERRNUM setting.

Example: Issuing Interface Environmental Commands

You can issue one or more Interface SET commands in a request using Direct SQL Passthru. Interface SET commands are not passed to the RDBMS; the Interface maintains them in memory.

The following example specifies the DB2 RDBMS as the target RDBMS, issues four Interface SET commands, and issues the SQL ? query command to display the updated parameters. Notice that the request does not include the environmental qualifiers TSO, MVS, or CMS.

```
set sqlengine=db2
 >  > sql set autoclose on fin
 >  > sql set ssid dsn
 >  > sql set plan p7009411
 >  > sql set dbspace public.space0
 >  > sql ?
(FOC1440) CURRENT SQL INTERFACE SETTINGS ARE :
(FOC1442) CALL ATTACH FACILITY IS             -  : ON
(FOC1447) SSID FOR CALL ATTACH IS             -  : DSN
(FOC1448) ACTIVE PLAN FOR CALL ATTACH IS      -  :
(FOC1459) USER SET PLAN FOR CALL ATTACH IS    -  :
(FOC1460) INSTALLATION DEFAULT PLAN     IS    -  : P7009701
(FOC1503) SQL STATIC OPTION IS                -  : OFF
(FOC1444) AUTOCLOSE OPTION IS                 -  : ON FIN
(FOC1496) AUTODISCONNECT OPTION IS            -  : ON FIN
(FOC1499) AUTOCOMMIT OPTION IS                -  : ON COMMAND
(FOC1446) DEFAULT DBSPACE IS                  -  :
(FOC1449) CURRENT SQLID IS                    -  : SYSTEM DEFAULT
(FOC1424) ISOLATION LEVEL FOR DB2 TABLE INTERFACE IS  :
(FOC1441) WRITE FUNCTIONALITY IS              -  : ON
(FOC1445) OPTIMIZATION OPTION IS              -  : ON
(FOC1484) SQL ERROR MESSAGE TYPE IS           -  : FOCUS
(FOC1497) SQL EXPLAIN OPTION IS               -  : OFF
(FOC1552) INTERFACE DEFAULT DATE TYPE         -  : NEW
 >  >
```

See Chapter 11, *Environmental Commands*, for Interface SET commands.

Example: Issuing Native SQL Commands (Non-SELECT)

When the Interface identifies SQL commands, it passes them to the RDBMS for immediate execution.

With Direct SQL Passthru, one SQL command can span several lines without any prefix or continuation characters. You must complete the command or request with the END keyword.

This example specifies the DB2 RDBMS as the target RDBMS, creates the DPBRANCH table, and inserts rows.

```
SET SQLENGINE=DB2

SQL CREATE TABLE DPBRANCH
    (BRANCH_NUMBER    SMALLINT     NOT NULL,
     BRANCH_NAME      CHAR(5)      NOT NULL,
     BRANCH_MANAGER   CHAR(5)      NOT NULL,
     BRANCH_CITY      CHAR(5)      NOT NULL)
 IN PUBLIC.SPACE0 ;
END

SQL INSERT INTO DPBRANCH VALUES (1,'WEST','PIAF','NY') ;
END

SQL INSERT INTO DPBRANCH VALUES (2,'EAST','SMITH','NY') ;
END
```

See Appendix B, *Native SQL Commands*, for information on native SQL commands. Consult Appendix C, *File Descriptions and Tables*, for sample tables.

Example: Issuing SQL SELECT Commands

When you issue an SQL SELECT request using the Direct SQL Passthru facility, the Interface passes the request directly to the RDBMS. FOCUS does not examine it. The RDBMS evaluates the request and sends storage requirements to FOCUS for future request results (answer sets).

Based on the storage requirements, FOCUS creates an internal Master File named SQLOUT (discussed in *The SQLOUT Master File* on page 9-8). The SQLOUT Master File functions as a template for reading and formatting the request results.

After FOCUS prepares the storage area and the SQLOUT Master File, the RDBMS executes the SQL SELECT request, retrieves the rows, and returns the answer set to FOCUS. The answer set is, in effect, a default report. FOCUS performs minimal additional formatting. When the report is complete, the internal SQLOUT Master File is discarded.

The following example illustrates a SELECT statement and its default report. The SELECT statement retrieves, from the inventory table, the total number of individual units of each vendor's products for all branches located in New York. The subquery retrieves New York branch numbers. The request does not specify additional report formatting:

```
SQL DB2
SELECT VENDOR_NUMBER, PRODUCT, SUM(NUMBER_OF_UNITS)
FROM DPINVENT
WHERE BRANCH_NUMBER IN
  (SELECT BRANCH_NUMBER
   FROM DPBRANCH
   WHERE BRANCH_CITY = 'NY')
GROUP BY VENDOR_NUMBER, PRODUCT
ORDER BY VENDOR_NUMBER, PRODUCT;
END
>
>
 NUMBER OF RECORDS IN TABLE=        2  LINES=      2

 PAGE     1

 VENDOR_NUMBER  PRODUCT  __SUM(NUMBER_OF_UNITS)
 -------------  -------  ----------------------
             1  RADIO                        15
             3  MICRO                         9
```

Internally, the process produces an SQLOUT Master File (described in *The SQLOUT Master File* on page 9-8) that you can use for FOCUS report formatting commands.

The SQLOUT Master File resides in memory; you cannot edit or print it. It exists only for the duration of the request, so subsequent requests cannot reference it. To create an internal Master File that exists for the entire FOCUS session, see *Creating FOCUS Views With Direct SQL Passthru* on page 9-12.

To produce the preceding default report, the Direct SQL Passthru facility implicitly issues the following TABLE request against the SQLOUT Master File:

```
TABLE FILE SQLOUT
PRINT *
END
```

The RDBMS reads data once for SELECT requests via Direct SQL Passthru. FOCUS does not hold or re-read data locally, except for some types of TABLE subcommands (for example, to re-sort or summarize rows). For performance reasons, you should incorporate as many operations as possible (particularly, sorting and aggregation operations) in your SELECT statement and rely on FOCUS for formatting and operations not available through the RDBMS.

For an example of a SELECT request that includes FOCUS report formatting commands, see *The SQLOUT Master File* on page 9-8.

The SQLOUT Master File

To give you access to FOCUS report formatting facilities, the Interface generates the SQLOUT Master File for each SQL SELECT query. The SQLOUT Master File supports read-only access. The Interface creates this internal Master File in memory based on information from the RDBMS. You cannot edit or print the SQLOUT Master File; it exists only for the immediate request, so subsequent requests cannot reference it.

Note: The Interface does not generate an associated Access File, since the SQL statement is stored in memory.

The SQLOUT Master File describes the answer set. Each field represents one data element in the outermost SELECT list, reflecting the flat row that the RDBMS returns. The following is the SQLOUT Master File created for the example in *Example: Issuing SQL SELECT Commands* on page 9-6:

```
FILENAME=SQLOUT, SUFFIX=SQLDS, $
  SEGNAME=SQLOUT, SEGTYPE=S0, $
    FIELD='VENDOR_NUMBER' , E01, USAGE=I6 ,ACTUAL=I2 ,MISSING=OFF, $
    FIELD='PRODUCT'       , E02, USAGE=A5 ,ACTUAL=A5 ,MISSING=OFF, $
    FIELD='__SUM(NUMBER_OF_UNITS)',
                            E03, USAGE=I9 ,ACTUAL=I4 ,MISSING=OFF, $
```

The FILENAME and the SEGNAME values are SQLOUT. The SUFFIX is SQLDS. The SEGTYPE is S0. These values are constant.

The Interface uses the RDBMS DESCRIBE function to obtain column information:

- The FIELDNAME value is the column name. For expressions or functions, the FIELDNAME value depends on whether the RDBMS is SQL/DS or DB2:
  - The SQL/DS RDBMS returns a column name that FOCUS uses as a fieldname.
  - The DB2 RDBMS does not return a column name, so the fieldname and the alias are assigned the same default values.
- The ALIAS value is set to E0n (n starts at 1 and is incremented by 1 for each data element in the outermost SELECT list).
- The MISSING value is ON if the column allows Nulls; otherwise, it is set to OFF.

The Interface calculates USAGE and ACTUAL formats based on the column datatype and length. The following chart outlines these calculations:

		ACTUAL Formats MISSING=	
Datatype:	**USAGE Format:**	**OFF**	**ON**
`DATE`	YYMD	DATE	same
`TIME`	A8	A8	same
`TIMESTAMP`	A26	A26	same
`INTEGER`	I9	I4	same
`SMALLINT`	I6	I2	I4
`DECIMAL`	Pp.s	P((p+1)/2)	P((p+1)/2) if p>15, P8 if p≤15
Where p is precision and s is scale. Precision in the USAGE format includes the decimal point and sign; precision in the ACTUAL format excludes them.			
`FLOAT` (4 byte)	F9.2	F4	same
`FLOAT` (8 byte)	D12.2	D8	same
`VARCHAR(n)`	An	An	same
Where n ≤ 254 characters.			
`VARCHAR(n)`	TX50	TX	same
Where 254 < n ≤ 4094 characters. **Note:** VARCHAR strings > 4094 characters are not supported.			
`CHAR(n)`	An	An	same
`GRAPHIC(n)`	A(2n+2)	Kn	same
Where n ≤ 127 characters.			
`VARGRAPHIC(n)`	A(2n+2)	Kn	same
Where n ≤ 127 characters **Note:** VARGRAPHIC(n) where n >127 characters is not supported.			

Note: Results of expressions are also one of these data types.

You can use the SET CONVERSION command to alter the length and scale of numeric columns displayed from a SELECT request. That is, you can control the USAGE attribute in the dynamically created Master File.

The syntax is

```
SQL [target_db] SET CONVERSION {RESET|datatype} [RESET|PRECISION {value|MAX}]
```

where:

target_db

Indicates the target RDBMS. Acceptable values are DB2 and SQLDS. Omit if you previously issued the SET SQLENGINE command.

RESET

Returns precision and scale values that you previously altered back to the Interface defaults. If you specify RESET immediately following the SET CONVERSION command, all datatypes return to the defaults. If you specify RESET following a particular data type, only columns of that data type are reset.

datatype

Applies the command only to columns of a specific datatype. Valid data types are:

INTEGER	INTEGER and SMALLINT.
DECIMAL	DECIMAL.
REAL	Single precision floating point.
FLOAT	Double precision floating point.

value

Is the precision in the following form:

nn [*mm*]

where:

nn	Must be greater than 1 and less than the maximum allowable value for the data type. (See description of MAX.)
mm	Is the scale. Valid with DECIMAL, REAL, and FLOAT datatypes. If you do not specify a value for scale, the current scale setting remains in effect.

MAX

Sets the precision to the maximum allowable value for the indicated data type:

DATA TYPE	**MAX Precision**
INTEGER	11
REAL	9
FLOAT	20
DECIMAL	33

Note: You must include space for the decimal point and for a negative sign (if applicable) in your precision setting.

For example, to set the precision for all INTEGER and SMALLINT fields to 7:

```
SQL DB2 SET CONVERSION INTEGER PRECISION 7
```

To set the precision for all DOUBLE PRECISION fields to 14 and the scale to 3:

```
SQL DB2 SET CONVERSION FLOAT PRECISION 14 3
```

To set the precision for all INTEGER and SMALLINT fields to the default:

```
SQL DB2 SET CONVERSION INTEGER RESET
```

To set the precision and scale for all fields to the defaults:

```
SQL DB2 SET CONVERSION RESET
```

The following example illustrates how to customize the default report output from the example in *Example: Issuing SQL SELECT Commands* on page 9-6. The example adds the TABLE FILE SQLOUT statement to the SELECT statement, and follows it by a report heading and AS phrases that rename column headings. The primary purpose of the TABLE FILE extension is for report formatting:

```
SQL DB2
  .
  .
  .
ORDER BY VENDOR_NUMBER, PRODUCT;
TABLE FILE SQLOUT
"Number of Units of Each Vendor's Products"
"         in New York Branches            "
" "
PRINT E01 AS 'Vendor,Number'
     E02 AS 'Product,Name'
     E03 AS 'Total,Units'
END
>
>
 NUMBER OF RECORDS IN TABLE=        2  LINES=      2

 PAGE     1

 Number of Units of Each Vendor's Products
          in New York Branches

 Vendor  Product      Total
 Number  Name         Units
 ------  -------      -----
      1  RADIO           15
      3  MICRO            9
```

When customizing a report, standard FOCUS report request syntax applies, subject to the following rules:

- You must specify fieldnames or aliases from the SQLOUT Master File.
- You may include any FOCUS TABLE formatting options or subcommands that you can code using the SQLOUT Master File.
- Most reporting operations are available. For example, you can use FOCUS direct operators, calculate COMPUTE fields, re-sort the answer set, reorder or suppress the printing of fields, or create extract files with various formats, including HOLD FORMAT SQL.
- DEFINE fields are not permitted. They are permitted for FOCUS views created with Direct SQL Passthru. See *Creating FOCUS Views With Direct SQL Passthru* on page 9-12.
- The ?F query command is not available for the SQLOUT Master File; it is available for FOCUS views created with Direct SQL Passthru. See *Creating FOCUS Views With Direct SQL Passthru* on page 9-12 for more information.

Creating FOCUS Views With Direct SQL Passthru

You can create a named, internal Master File (FOCUS view) for a particular SELECT statement with the Interface SQL PREPARE command. Unlike the SQLOUT Master File, you can generate reports with this FOCUS view for the entire FOCUS session.

In any FOCUS session, you can describe an unlimited number of FOCUS views. The syntax is

```
SQL [target_db] PREPARE view_name FOR
SELECT....[;]
END
```

where:

target_db
: Indicates the target RDBMS. Acceptable values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

view_name
: Names the Master File (FOCUS view). The name can be eight characters long and must conform to FOCUS naming conventions for Master Files (see Chapter 4, *Describing Tables to FOCUS*).

SELECT...
: Is any SELECT statement.

Note:

- You cannot include FOCUS formatting options or subcommands. Issue a TABLE request against the view name to create a report.

- FOCUS views created with SQL PREPARE provide read-only access to data; write operations are not permitted.
- Do not confuse creating a FOCUS view with creating a DB2 view.

The keywords and default aliases in the generated Master File are the same as those for the SQLOUT Master File (see *The SQLOUT Master File* on page 9-8). Only its file or member name and the FILENAME value reflect the view name you specify in the PREPARE command.

With Direct SQL Passthru, the PREPARE command only creates an internal Master File. The RDBMS does not return data until you execute a TABLE request referencing the FOCUS view. PREPARE does not generate an Access File; you supply the table names in SELECT statement FROM clauses.

Master Files created with PREPARE reside in memory, so you cannot edit or print them. They function like any other FOCUS Master File; for example, you can specify them with TableTalk. You can assign DEFINE fields to them or use them in MATCH FILE commands. You can also issue the ?F query in TABLE requests to list the fieldnames of the FOCUS view.

In this example, the DB2 RDBMS is the target RDBMS. The SQL PREPARE command creates a view named TOTPROD:

```
SET SQLENGINE=DB2
>
SQL PREPARE TOTPROD FOR
SELECT VENDOR_NUMBER, PRODUCT, SUM(NUMBER_OF_UNITS)
FROM DPINVENT
WHERE BRANCH_NUMBER IN
  (SELECT BRANCH_NUMBER
   FROM DPBRANCH
  WHERE BRANCH_CITY = 'NY')
GROUP BY VENDOR_NUMBER, PRODUCT
ORDER BY VENDOR_NUMBER, PRODUCT;
END
```

After the TOTPROD view is created, you can assign a temporary HI_STOCK field to the TOTPROD Master File and specify the temporary field in a report request:

```
>
DEFINE FILE TOTPROD
  HI_STOCK/A2 = IF (E03 GT 10) THEN '**' ELSE '  ' ;
END
>
TABLE FILE TOTPROD
"Number of Units of Each Vendor's Products"
"         in New York Branches            "
" "
PRINT E01 AS 'Vendor,Number'
      E02 AS 'Product,Name'
      E03 AS 'Total,Units'
      HI_STOCK AS ''
FOOTING
"** = Too many units in stock,"
"     reevaluate purchasing. "
END
>

 NUMBER OF RECORDS IN TABLE=        2  LINES=      2

 PAGE     1

 Number of Units of Each Vendor's Products
          in New York Branches

 Vendor  Product      Total
 Number  Name         Units
 ------  -------      -----
      1  RADIO           15  **
      3  MICRO            9

 ** = Too many units in stock,
      reevaluate purchasing.
```

The one restriction on FOCUS views created with Direct SQL Passthru involves using them with the FOCUS JOIN command. You cannot join two FOCUS views or a view with another FOCUS-readable source. You can, however, create a HOLD file of data extracted from the FOCUS view and use the HOLD file in the join.

Parameterized SQL Passthru

The Direct SQL Passthru facility supports parameterized SQL statements. These statements incorporate parameter markers to indicate where a value should be substituted, so you can execute the SQL statements multiple times with varying input values.

The following is an example of a parameterized SQL statement:

```
INSERT INTO INVENTORY (PARTNO) VALUES(?)
```

The INSERT statement is executed once for each value you provide for the parameter marker (?), and a new row with that value is placed in the PARTNO column.

Parameterized SQL Passthru provides the following advantages:

- You can execute an SQL statement using varying values without keying in the entire SQL statement for each value.
- Internally, it utilizes the optimal SQL for the native RDBMS.
- The execution of a FOCUS session becomes equivalent to that of a 3GL program, with a framework consisting of such elements as compiled statements and parameter markers (input values).

Note:

- All errors are posted to Dialogue Manager variables &RETCODE and &FOCERRNUM.
- Direct SQL Passthru and Parameterized SQL Passthru are not available within MODIFY.

Parameterized SQL Command Summary

With parameterized SQL you can compile, bind, and repeatedly execute a series of SQL commands. To avoid invoking END processing between the SQL statements in the series, you place the whole sequence of SQL requests within SQL BEGIN SESSION and SQL END SESSION commands.

To incorporate parameter markers in SQL statements, first compile the statements with the PREPARE command, then bind them with the BIND command, and subsequently execute them with the EXECUTE command. Place this group of actions within a BEGIN SESSION/END SESSION pair. You can also include other FOCUS, SQL, and Interface environmental commands within the BEGIN SESSION/END SESSION pair.

Subsequent sections explain the individual commands involved in Parameterized Passthru design. *Sample Session* on page 9-23 contains a sample session. The general syntax is

```
SQL [target_db] command [;]
[TABLE FILE statement_name]
[options]
END
```

where:

target_db

Indicates the target RDBMS. Acceptable values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

command

Is one of the following: BEGIN SESSSION, END SESSION, PREPARE, BIND, or EXECUTE.

;

For SQL SELECT requests only, the semicolon is required if you intend to specify additional FOCUS report options.

TABLE FILE

Is permitted only with PREPARE and EXECUTE commands that invoke SQL SELECT requests; invokes FOCUS report formatting options or operations. By including a TABLE FILE request, you can produce different customized reports with one SQL query. The answer set is returned at EXECUTE time. If you include a TABLE FILE request in both a PREPARE and EXECUTE command for the same SQL statement, the EXECUTE request takes precedence.

statement_name

Is the name of a PREPAREd SELECT statement.

options

Are FOCUS report formatting options.

END

Terminates the request. Is optional for Interface SET commands, the SQL commands COMMIT WORK and ROLLBACK WORK, the DB2 CONNECT command, and the Interface parameterized Passthru commands BEGIN SESSION, END SESSION, and PURGE (discussed in subsequent sections). Required for all other commands.

Note: Do not confuse the SQL PREPARE and BIND statements with the RDBMS prepare and bind; the RDBMS versions are not available through the Interface.

BEGIN/END SESSION

The BEGIN SESSION command begins a sequence of Direct SQL Passthru commands; the END SESSION command terminates the sequence. The syntax is

```
SQL [target_db] {BEGIN|END} SESSION
```

where:

target_db

Indicates the target RDBMS. Acceptable values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

`BEGIN`

Indicates that a sequence of SQL commands is to be passed to the RDBMS. While the BEGIN option is in effect, the END syntax that terminates each Direct SQL Passthru statement does not automatically release resources.

`END`

Indicates the end of a sequence of SQL commands. Closes all cursors and releases all resources. Executes actions specified in SET action ON COMMAND (see Chapter 10, *Thread Control Commands*). Purges all statements PREPAREd inside the BEGIN SESSION/END SESSION pair.

After the END SESSION command is executed, cursors and statements PREPAREd within the BEGIN SESSION/END SESSION pair are unavailable. The sequence of statements within the BEGIN SESSION/END SESSION pair can include:

- SQL Passthru commands PREPARE, BIND, and EXECUTE.
- FOCUS commands that refer to Direct SQL Passthru-created views.
- Other FOCUS commands such as TABLE and MODIFY against SQL or other files.

  Note:

  - FIN issues END COMMAND and then purges all statements PREPAREd outside the BEGIN SESSION/END SESSION pair.
  - FOCUS commands must use the same DB2 plan.
- SQL commands. The SQL commands COMMIT WORK and ROLLBACK WORK are particularly useful in Parameterized SQL Passthru requests.
- Environmental commands such as SET PLAN, SET SSID, DB2 CONNECT, and SQL/DS CONNECT statements.

If you omit the BEGIN SESSION/END SESSION pair, the Interface automatically brackets each individual Direct SQL Passthru command with BEGIN SESSION and END SESSION. The execution of the END SESSION (either implicitly or explicitly) in a Direct SQL Passthru statement invokes actions requested in SET action ON COMMAND (see Chapter 10, *Thread Control Commands*). You can use statements PREPAREd without an explicit BEGIN SESSION/END SESSION pair in TABLE requests, but you cannot use them in the EXECUTE statement.

COMMIT WORK

COMMIT WORK terminates a unit of work and makes all database changes permanent. The syntax is

```
SQL [target_db] COMMIT WORK
```

where:

target_db
: Indicates the target RDBMS. Acceptable values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

After execution of the COMMIT WORK command, the RDBMS drops the PREPAREd status of SQL statements; it also releases locks. If you need a PREPAREd version of the statement, you must issue the SQL PREPARE statement again.

ROLLBACK WORK

ROLLBACK WORK terminates a unit of work and restores all data changed by SQL statements to their state at the last commit point. The syntax is

```
SQL [target_db] ROLLBACK WORK
```

where:

target_db
: Indicates the target RDBMS. Acceptable values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

After execution of the ROLLBACK WORK command, the RDBMS drops the PREPAREd status of SQL statements. If you need a PREPAREd version of the statement, you must issue the SQL PREPARE statement again.

PREPARE

The PREPARE command PREPAREs (checks syntax, then compiles) an SQL statement and stores the compiled version for later use. The SQL statement can contain parameter markers. The syntax is

```
SQL [target_db] PREPARE statement_name FOR sql_statement [;]
```

where:

target_db

Indicates the target RDBMS. Acceptable values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

statement_name

Is the 1-to 8-character name of an SQL variable that will contain the PREPAREd (compiled) version of an SQL statement.

sql_statement

Is a character-string expression that can include parameter markers; it represents the SQL statement to PREPARE. The statement must be one of the following:

- UPDATE (WHERE CURRENT OF CURSOR is not supported)
- DELETE (WHERE CURRENT OF CURSOR is not supported)
- INSERT
- SELECT (Master File is created to represent the returned answer set)
- CREATE
- DROP
- ALTER
- COMMENT
- LABEL
- GRANT
- REVOKE
- COMMIT
- ROLLBACK
- LOCK
- EXPLAIN

;

Is required if sql-statement is a SELECT statement followed by TABLE FILE report options.

For example, consider the following SQL PREPARE command:

```
SQL DB2 PREPARE D FOR DELETE FROM USER1.EMPLOYEE
   WHERE  EMP_ID = ?
```

Variable D will contain the PREPAREd version of the following SQL string:

```
DELETE FROM USER1.EMPLOYEE WHERE EMP_ID = ?
```

You supply values for the parameter marker in the statement by issuing an EXECUTE statement for variable D. The parameter marker allows you to execute the same DELETE statement many times with different values of EMP_ID. You can use a parameter marker anywhere a literal value appears in an SQL statement.

Note:

- Answer sets for FOCUS TABLE requests in PREPAREd SELECT statements are returned at EXECUTE time even if they were specified in the PREPARE statement.

  If both the PREPARE and EXECUTE commands specify a TABLE FILE request for the same statement, the EXECUTE request takes precedence.

- PREPARE for an already PREPAREd statement unprepares the statement by means of PURGE. As a side effect, BIND for this statement (if any) is cleared.

EXECUTE

This statement executes a previously PREPAREd SQL statement. If the SQL statement includes parameter markers, you must supply their values in the USING clause. If the SQL statement is a SELECT statement, you can use the TABLE FILE extension to produce a formatted report.

The syntax is

```
SQL [target_db] EXECUTE statement_name [USING data_list] [;]
```

where:

target_db
: Indicates the target RDBMS. Acceptable values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

statement_name
: Is the 1-to 8-character name of an SQL variable that contains the PREPAREd (compiled) version of the SQL statement to execute. Use this name in a TABLE FILE extension if you want a formatted report.

data_list
: Is a list of arguments to substitute for the parameter markers in the PREPAREd SQL statement. Separate arguments in the list with commas.

;
: Is required if the PREPAREd SQL statement is a SELECT statement followed by TABLE FILE report options.

For example, to execute the PREPARE statement in the previous example, issue:

```
SQL DB2 EXECUTE D USING 'AAA';
```

The DELETE statement from *PREPARE* on page 9-19 is sent to the DB2 RDBMS; the RDBMS deletes all rows with an EMP_ID value of 'AAA' from the database. The resulting SQLCODE is returned to the program.

You can execute this statement many times within the same unit of recovery by supplying different values for the EMP_ID to delete.

The SQL commands COMMIT WORK and ROLLBACK WORK destroy all statements PREPAREd in a unit of recovery. Thus, after a COMMIT or ROLLBACK, you must again PREPARE any statement you want to EXECUTE.

Note:

- Use of EXECUTE outside a BEGIN SESSION/END SESSION pair produces fatal error FOC1477, “Invalid use of BIND or EXECUTE”.
- Use of EXECUTE for a statement PREPAREd outside a BEGIN SESSION/END SESSION pair produces fatal error FOC1477, “Invalid use of BIND or EXECUTE”.
- Answer sets for FOCUS TABLE requests in compiled SELECT statements are returned at EXECUTE time even if they were specified in the PREPARE statement.

  If both the PREPARE and EXECUTE commands specify a TABLE FILE request for the same statement, the EXECUTE request takes precedence.
- To specify missing values in an EXECUTE statement, you can use the word NULL or denote the missing column values by a comma. For example

  ```
  EXECUTE xyz USING 8,9,NULL,,'abcd'
  ```

PURGE

The PURGE command clears results from a previously issued PREPARE or BIND command. It is optional. The syntax is

```
SQL [target_db] PURGE statement_name [;]
```

where:

target_db
: Indicates the target RDBMS. Acceptable values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

statement_name
: Is the 1-to 8-character name of an SQL variable that contains a PREPAREd (compiled) version of an SQL statement.

For example:

```
SQL DB2 PREPARE D FOR DELETE FROM USER1.EMPLOYEE
    WHERE  EMP_ID = ?

SQL DB2 PURGE D;
```

BIND

You can use the BIND command to define the format of each parameter specified in a PREPARE command. The list of formats is comma delimited; each element is a datatype supported by the RDBMS. The syntax is

```
SQL [target_db] BIND statement_name USING format_list;
```

where:

target_db

Indicates the target RDBMS. Acceptable values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

statement_name

Is the 1-to 8-character name of an SQL variable that contains a PREPAREd (compiled) version of an SQL statement.

format_list

Is a comma-delimited list of datatypes used in the request. The following datatypes are supported by the RDBMS:

- SMALLINT
- INTEGER
- DECIMAL(m,n)
- FLOAT
- REAL
- DOUBLE
- VARCHAR(n)
- LONGVARCHAR(n)
- CHARACTER(n)
- DATE

Note:

- If a statement is PREPAREd and EXECUTEd without a corresponding BIND, the Interface uses default formats based on the RDBMS storage formats it detects for the columns referenced by parameter markers in the statement.
- Use of BIND outside a BEGIN SESSION/END SESSION pair produces fatal error FOC1477, “Invalid use of BIND or EXECUTE”.
- BIND without parameters clears a previous BIND for the statement.

  BIND is also cleared when you issue PREPARE for an already PREPAREd statement.

Sample Session

The following sample session illustrates the design of a Parameterized SQL application:

```
SQL DB2 BEGIN SESSION

SQL DB2 PREPARE ABC FOR UPDATE STARS SET NAME=? WHERE DISTANCE=? ;
END

SQL DB2 PREPARE DEF FOR SELECT * FROM STARS WHERE DISTANCE=? AND DENSITY=?;
END

SQL DB2 BIND ABC USING CHAR(6),DECIMAL(5,0);
END

SQL DB2 BIND DEF USING DECIMAL(5,0),DECIMAL(6,2);
END

-* repeat with different input data...
   SQL DB2 EXECUTE ABC USING 'GAMMA',555. ;
   END

   SQL DB2 EXECUTE ABC USING 'DELTA',777. ;
   END

   SQL DB2 EXECUTE ABC USING 'ALPHA',9640. ;
   END

   SQL DB2 EXECUTE DEF USING 555.,324.27; TABLE FILE DEF PRINT *
   END
-* end repeat

SQL DB2 COMMIT WORK ;
END

SQL DB2 END SESSION
```

Notice that the BIND commands provide formats and the EXECUTE commands provide values for the parameter markers. Since DEF represents a SELECT statement, the EXECUTE command for DEF can include a TABLE FILE DEF request. Alternatively, the TABLE FILE DEF request could have been included in the PREPARE DEF command instead of in the EXECUTE DEF command.

10 Thread Control Commands

Topics:

The Interface includes a variety of COMMIT, connection, and thread control capabilities. The SET AUTOaction ON event command implements these facilities.

This chapter includes:

- The SET AUTOaction ON event command. See *The SET AUTOaction ON Event Command* on page 10-1.
- The effects of various action and event combinations. See *Action and Event Combinations* on page 10-5.
- Examples of three types of sessions: default, user controlled, and pseudo-conversational, made possible by varying the settings. See *Combinations of SET AUTOaction Commands* on page 10-8.

The SET AUTOaction ON Event Command

Actions are RDBMS commands, such as COMMIT WORK, that the Interface issues in response to events in a FOCUS session. Events include the end of a MODIFY or TABLE request, interaction with the terminal, and the end of a FOCUS session.

For DB2, you can make a MODIFY transaction pseudo-conversational by issuing CLOSE and DISCONNECT automatically at all COMMIT points within the MODIFY procedure. Or, by delaying COMMITs until a group of commands is issued, you can combine FOCUS commands and native SQL commands in one Logical Unit of Work.

The SET AUTOaction command allows you to control when the Interface issues actions. Subsequent sections discuss each action. The syntax is

```
SQL [target_db] SET action ON event
```

where:

target_db

Indicates the target RDBMS. Valid values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command or to set AUTOCOMMIT ON CRTFORM in a MODIFY procedure.

action

Indicates the action to be taken by the Interface. Valid values are as follows:

`AUTOCOMMIT`	Issues the SQL command COMMIT WORK.
`AUTOCLOSE`	Applies to DB2 only. Is the DB2 Call Attachment Facility (CAF) CLOSE operation.
`AUTODISCONNECT`	Severs the connection between the RDBMS and the user's address space or virtual machine.

event

Is the event that triggers the action. Valid values are as follows:

`CRTFORM`	Is valid only in MODIFY procedures with the AUTOCOMMIT action. Issues a COMMIT before each interaction with the user's terminal. At the end of the MODIFY, the event setting reverts to its value prior to the AUTOCOMMIT ON CRTFORM. Omit target_db from the command. Refer to Chapter 12, *Maintaining Tables With FOCUS*, for more information on MODIFY.
`COMMAND`	Executes the specified action at the end of a MODIFY procedure, a TABLE request, a Direct SQL Passthru request, or a CALLDB2 subroutine containing embedded SQL. **Note:** The Interface does not generate an end-of-MODIFY COMMIT if there is no open Logical Unit of Work.
`FIN`	Executes the specified action automatically only after the FOCUS session has been terminated by the FOCUS FIN command. To execute the action within the session, you must issue it explicitly. Since you cannot issue CLOSE and DISCONNECT explicitly, this applies to COMMIT only.
`COMMIT`	Executes the specified action whenever a COMMIT or ROLLBACK is issued either as a native SQL command or because of a current AUTOCOMMIT setting.

Note:

- Depending on how often the event occurs (and the corresponding command is issued), the AUTOaction setting may result in considerable overhead. Almost none of this overhead is FOCUS related; it is MVS and DB2 or CMS and SQL/DS related.
- For DB2, you can use the CLOSE and DISCONNECT commands only if the DB2 Interface is installed to use CAF.
- All settings are session level and can be issued from either the FOCUS command line or a FOCEXEC, except AUTOCOMMIT ON CRTFORM, which can only be issued in a MODIFY procedure. In a MODIFY procedure, the only valid command is AUTOCOMMIT ON CRTFORM.
- You can issue all settings except AUTOCOMMIT ON CRTFORM in batch jobs.
- You can use the CLOSE and DISCONNECT actions with FOCUS running under either TSO or Information Builder's Multi-Session Option (MSO). Under MSO, the MSO address space connects to DB2 when the first MSO user (sub-task) requires DB2 services. This is followed by a less expensive connection for the sub-task, and a thread is opened to the application plan.
- If AUTOCOMMIT is set on COMMAND, the Interface issues a COMMIT WORK at the end of a MODIFY procedure provided there is an open Logical Unit of Work (LUW).

Actions

Actions control lock retention and the user's connection to the RDBMS.

AUTOCOMMIT

SET AUTOCOMMIT issues an SQL COMMIT WORK each time the specified event occurs. Until a COMMIT WORK, changes to the target database are conditional and locks are held on the affected data. Other users may have their work delayed waiting for locks to be released.

COMMIT WORK completes the changes to the database and releases locks, improving concurrency. On the other hand, delaying the COMMIT preserves the integrity of processed data through several FOCUS commands or an entire FOCUS session.

AUTOCLOSE

SET AUTOCLOSE initiates the DB2 Call Attachment Facility (CAF) CLOSE operation. It determines how long a thread (the connection between the application program in the user's address space and the DB2 application plan) is open. The thread is not the same as the address space connection to DB2; the AUTODISCONNECT setting, discussed in a subsequent section, controls that connection.

In FOCUS, an application program is one of the following:

- The dynamic FOCUS DB2 Interface.
- A TABLE or MODIFY procedure using static SQL.
- A CALLDB2 subroutine using embedded SQL.

Generally speaking, each program has a corresponding plan. (*Plan Management* in Chapter 13, *Static SQL*, includes a discussion of plan management.)

A site that installs a DB2 subsystem determines the maximum number of concurrent users (threads) the subsystem will support. Since each user requires enough virtual storage for his application plan, this setting controls the amount of storage the site wants to allocate to active DB2 users at any one time.

The CAF CLOSE command deallocates the DB2 thread, releasing the virtual storage for the application plan. DB2 requires that an existing thread to a plan be closed before a thread to another plan is opened. If a thread is closed without a subsequent OPEN operation, the closed thread becomes inactive; the user is still connected to DB2, but not to a particular application plan. The user (task) still owns the thread; it is not available to other users. To release the thread, the user must disconnect completely from DB2.

Note: AUTOCLOSE is ignored for SQL/DS.

AUTODISCONNECT

For DB2, DISCONNECT completely detaches the user's address space (or task) from DB2. This differs from CLOSE because, after a CLOSE, the FOCUS task is still connected to the DB2 subsystem and can open a thread to another plan. After a DISCONNECT, the FOCUS task must reestablish its connection to DB2 before doing any database work. FOCUS tasks that frequently issue the DISCONNECT command are connected to DB2 for shorter periods of time, allowing other tasks to connect and acquire threads as needed. However, there is significant system overhead associated with frequently connecting and disconnecting, and the possibility exists that no thread will be immediately available when the task attempts to reconnect.

For SQL/DS, DISCONNECT sponsors a COMMIT WORK RELEASE, disconnecting the user's virtual machine from the SQL/DS database machine and freeing a small amount of virtual storage in the database machine.

Action and Event Combinations

The following table summarizes possible combinations of actions and events:

- D indicates a default combination.
- X indicates a combination that either is not supported or does not apply.
- S indicates a supported combination.

Actions	**Events**			
	COMMIT	COMMAND	CRTFORM	FIN
COMMIT	X	D	S*	S
CLOSE (DB2 CAF only)	S	S	X	D
DISCONNECT	S	S	X	D

Note:

* Supported within a MODIFY procedure only.

Effects of Action and Event Combinations

This section discusses the advantages and disadvantages of certain combinations of actions and events.

SET AUTOCOMMIT ON CRTFORM

This command works only from within a MODIFY procedure and requires a slightly different syntax:

```
SQL SET AUTOCOMMIT ON CRTFORM
```

Note the absence of the target database qualifier after the SQL keyword. Including a qualifier generates a syntax error.

This is the “COMMIT as often as possible” strategy. FOCUS issues a COMMIT prior to displaying each CRTFORM, thus releasing all locks before presenting data to the user. AUTOCOMMIT ON CRTFORM invokes Change Verify Protocol (CVP); you must check the FOCURRENT variable to determine whether CVP detected a conflict with another user. *AUTOCOMMIT ON CRTFORM* in Chapter 12, *Maintaining Tables With FOCUS*, discusses this setting and some limits on its use. This setting also requires the KEYS value in the Access File to be greater than zero (see *Access Files* in Chapter 4, *Describing Tables to FOCUS*).

With this strategy, more concurrent users can access the same DB2 data. However, a COMMIT carries overhead and may be unnecessary for the application.

At the end of the MODIFY procedure, the event setting reverts to its value before the AUTOCOMMIT ON CRTFORM was issued.

SET AUTOCOMMIT ON FIN

During the FOCUS session, only those COMMIT and ROLLBACK commands you issue explicitly are executed. The Interface automatically issues a COMMIT at the end of the FOCUS session. The syntax is

```
SQL [target_db] SET AUTOCOMMIT ON FIN
```

where:

target_db
: Indicates the target RDBMS. Valid values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

With this setting, you do not incur COMMIT overhead until the end of the FOCUS session. You can also use it to suspend the COMMIT action for a period of time. See *Example: Explicit Control of a Logical Unit of Work (LUW)* on page 10-12 for an example.

MAINTAIN does not support the AUTOCOMMIT ON FIN setting.

Holding unneeded locks can cause contention in the DB2 or SQL/DS system and make other users wait for data. Delaying the COMMIT also puts data changes at risk in case of system or machine failure.

SET AUTOCOMMIT ON FIN gives you full control of Logical Units of Work (LUWs) within your FOCUS session. FOCUS relies on your explicit COMMIT and ROLLBACK commands. No implicit COMMIT or ROLLBACK is ever forced until the end of the FOCUS session (FIN command). Use this option cautiously to avoid possible locking problems and unpredictable consequences in cases of conflict with other AUTOaction settings.

Note: With AUTOCOMMIT ON FIN, SQL errors do not trigger an automatic ROLLBACK by the Interface. Examine each return code and take appropriate action to prevent unwanted changes from being committed at FIN.

SET AUTOCLOSE ON COMMAND

This setting closes the DB2 thread at the end of each command. The syntax is:

```
SQL [DB2] SET AUTOCLOSE ON COMMAND
```

Omit the DB2 qualifier if you previously issued the SET SQLENGINE command.

This setting releases some virtual storage, but there is a cost to repeatedly initializing a thread. Also, closing a thread does not make it available to other users; only a disconnect can release it.

Note: AUTOCLOSE is ignored for SQL/DS. See the description of AUTODISCONNECT for a comparable function.

SET AUTODISCONNECT ON COMMIT

For DB2, when a COMMIT is issued, the CAF facility disconnects the FOCUS session from DB2, terminating the DB2 thread. For SQL/DS, the virtual machine disconnects from the RDBMS after each COMMIT. The syntax is

```
SQL [target_db] SET AUTODISCONNECT ON COMMIT
```

where:

target_db
: Indicates the target RDBMS. Valid values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

This setting frees the DB2 thread for use by other users. The disadvantage is the cost of repeatedly connecting to DB2 and acquiring a thread. Threads, once released, may not be available when needed, so you may experience delays while your request waits for a thread.

Note: When you combine this setting with AUTOCOMMIT ON FIN, you can control the span of the DB2 session.

SET AUTOCLOSE ON FIN

The Interface issues a CAF CLOSE at the end of the FOCUS session. The syntax is

```
SQL [DB2] SET AUTOCLOSE ON FIN
```

Omit the DB2 qualifier if you previously issued the SET SQLENGINE command.

This setting duplicates Interface default behavior, unless the Interface was installed with AUTOCLOSE ON COMMAND as the default.

Note: AUTOCLOSE is ignored for SQL/DS. See the description of AUTODISCONNECT for a comparable function.

SET AUTODISCONNECT ON FIN

This command disconnects FOCUS from the RDBMS at the end of the FOCUS session, duplicating Interface default behavior. The syntax is

```
SQL [target_db] SET AUTODISCONNECT ON FIN
```

where:

target_db
: Indicates the target RDBMS. Valid values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

Combinations of SET AUTOaction Commands

Think of actions as a nested sequence; the COMMIT action is the innermost level, then CLOSE, then DISCONNECT. Events are also organized hierarchically; CRTFORM is the innermost level, then COMMIT, then COMMAND, and, finally, FIN. In general, you should not set the outer of two actions to occur more frequently than the inner action. Recommended combinations follow:

```
When AUTOCOMMIT     is set to FIN,
     AUTOCLOSE      must be   FIN,
     AUTODISCONNECT must be   FIN*.

When AUTOCLOSE      is set to FIN
     AUTODISCONNECT must be   FIN.

When AUTOCOMMIT     is set to COMMAND or CRTFORM,
     AUTOCLOSE      may be    FIN, COMMAND or COMMIT,
     AUTODISCONNECT may be    FIN, COMMAND or COMMIT.

When AUTOCLOSE      is set to COMMAND or COMMIT,
     AUTODISCONNECT may be    FIN, COMMAND or COMMIT.

When AUTOCLOSE or AUTODISCONNECT is set to COMMIT,
     AUTOCOMMIT     may be    COMMAND or CRTFORM.

* Violations of this rule can cause unpredictable behavior.
```

DB2 Note:

- Whenever you open a thread to a new application plan or connect to a new DB2 subsystem, the Interface issues a COMMIT WORK regardless of the AUTOCOMMIT setting.
- If the event that triggers AUTODISCONNECT occurs more frequently than the event that triggers AUTOCLOSE, the Interface forces a CAF CLOSE prior to any CAF DISCONNECT.
- In order to use the CLOSE and DISCONNECT commands, the DB2 Interface must be installed to use CAF.

Types of FOCUS Sessions

You can establish the following three types of FOCUS sessions by varying the combinations of SET AUTOaction ON event commands. Each is illustrated in a subsequent section:

- The default Interface session uses the settings AUTOCOMMIT ON COMMAND, AUTOCLOSE ON FIN, and AUTODISCONNECT ON FIN.

- The user-controlled session uses the settings AUTOCOMMIT ON FIN, AUTOCLOSE ON FIN, and AUTODISCONNECT ON FIN.
- The pseudo-conversational session uses the settings AUTOCOMMIT ON CRTFORM, AUTOCLOSE ON COMMIT, and AUTODISCONNECT ON COMMIT.

The Default Interface Session

The following illustration shows the duration of connections, threads and Logical Units of Work (LUWs) using the default Interface settings:

```
                      connection
      |--------------------------------------------------------->|
                      thread
      |--------------------------------------------------------->|


      |------LUW----->|--- LUW ------>|      |--- LUW ----->|
                                                                 F
 | . . . . . . F O C U S . . . s e s s i o n . . . . . . . . . . I
                                                                 N
      |               |               |      |              |
      cmd           user            cmd      cmd           cmd
      beg          COMMIT           end      beg           end
      |--------- MODIFY ------------->|      |----TABLE --->|

(a)  (b)             (c)             (d)    (e)            (f)  (g)

                                     AUTOCOMMIT ON COMMAND
                                     AUTOCLOSE ON FIN
                                     AUTODISCONNECT ON FIN
```

The FOCUS session begins at point (a) and ends at point (g), with the FIN command. The DB2 connection and thread are established at point (b), the first MODIFY call to DB2, and are retained for the entire FOCUS session.

The LUW initiated at point (b) by the MODIFY is terminated by the explicit COMMIT issued at point (c) within the MODIFY. Point (c) also marks the start of the next LUW which is retained until the end of the MODIFY command at point (d), where the Interface automatically generates a COMMIT.

A third LUW begins for the TABLE request. It is terminated by the COMMIT automatically generated at the end of the TABLE command. At FIN, the Interface closes the thread and terminates the connection to DB2. No COMMIT is generated at FIN.

The User-Controlled Session

The following illustration shows a session in which the user controls the duration of each Logical Unit of Work (LUW). The Interface does not automatically issue any COMMIT, CLOSE, or DISCONNECT command until after the FIN at the end of the FOCUS session:

```
                      connection
      |------------------------------------------------------------->|
                      thread
      |------------------------------------------------------------->|

      |----------LUW------------------------>|        |--LUW-->|
                                                                     F
 | . . . . . . F O C U S . . . s e s s i o n . . . . . . . . . . . . I
                                                                     N
      |             |      |           |     |        |        |
      cmd         cmd      cmd       cmd   user       cmd    cmd
      beg         end      beg       end  COMMIT      beg    end
      |----TABLE--->|      |-PASSTHRU->|              |CALLDB2>|

(a)  (b)           (c)    (d)         (e)   (f)      (g)      (h)   (i)

                                      AUTOCOMMIT ON FIN
                                      AUTOCLOSE ON FIN
                                      AUTODISCONNECT ON FIN
```

The FOCUS session begins at point (a) and ends at point (i) with the FIN command.

The connection and thread are established by the TABLE command at point (b) and retained until FIN. The LUW also starts at point (b). It is completed at point (f) when the user issues COMMIT WORK as an SQL Passthru request.

The final LUW, triggered by the CALLDB2 subroutine, is not terminated until FIN since there is no other user-issued COMMIT.

The Pseudo-Conversational Session

The next illustration shows how connections and threads can be dropped and re-established within a MODIFY procedure. The AUTOCOMMIT ON CRTFORM, issued within the MODIFY, automatically generates a COMMIT whenever a CRTFORM is displayed:

```
      |--connection--->|--connection->|-conn-->|--connection-->|

      |--thread------->|--thread----->|-thd--->|--thread------>|

      |--LUW---------->|--LUW-------->|-LUW--->|--LUW--------->|
                                                                     F
 | . . . . . . F O C U S . . . s e s s i o n . . . . . . . . . . . . I
                                                                     N
      |                |              |        |               |
      cmd              CRTFORM       user      CRTFORM       cmd
      beg                           COMMIT                   end
      |---------------------MODIFY---------------------------->|
(a)  (b)               (c)            (d)     (e)             (f)   (g)

                                      AUTOCOMMIT ON CRTFORM
                                      AUTOCLOSE ON COMMIT
                                      AUTODISCONNECT ON COMMIT
```

The FOCUS session begins at point (a) and ends at point (g) with the FIN command.

The connection, thread and Logical Unit of Work (LUW) are established at point (b), the beginning of the MODIFY procedure. The first CRTFORM triggers a COMMIT, terminating the LUW, thread, and connection. The Interface automatically re-establishes them all at the next call to DB2, and all are terminated by the user-issued COMMIT at point (d).

The process is repeated two more times; at the end of the MODIFY procedure, point (f), AUTOCOMMIT reverts to its previous (default) COMMAND setting, and the Interface generates a COMMIT.

Example: Explicit Control of a Logical Unit of Work (LUW)

In prior releases, the Interface automatically followed each native SQL request with a COMMIT; therefore, you could not have more than one native SQL command in a single LUW.

The following example demonstrates how to explicitly control the scope of an LUW using the expanded AUTO command settings. Numbers to the left refer to explanatory notes that follow the example.

Dialogue Manager control statements govern COMMIT or ROLLBACK processing based on the &RETCODE value. The example treats any value other than 0 as a failure:

```
    -TOP
    SQL DB2 SET AUTODISCONNECT ON FIN
    SQL DB2 SET AUTOCLOSE ON FIN
1.  SQL DB2 SET AUTOCOMMIT ON FIN

2.  SQL DB2 LOCK TABLE XYZ IN EXCLUSIVE MODE
    END
3.  -IF &RETCODE NE 0 GOTO ROLLBACK ;

4.  SQL DB2 INSERT INTO XYZ VALUES ('A','B','C','D') ;
    END
    -IF &RETCODE NE 0 GOTO ROLLBACK ;

    SQL DB2 INSERT INTO XYZ VALUES ('E','F','G','H') ;
    END
    -IF &RETCODE NE 0 GOTO ROLLBACK ;

    SQL DB2 INSERT INTO XYZ VALUES ('I','J','K','L') ;
    END
    -IF &RETCODE NE 0 GOTO ROLLBACK ;

5.  SQL DB2 COMMIT WORK;
    END
    -IF &RETCODE NE 0 GOTO ROLLBACK ;

    -GOTO OUT

    -ROLLBACK
    SQL DB2 ROLLBACK WORK;
    END

    -OUT
6.  SQL DB2 SET AUTOCOMMIT ON COMMAND
```

Note:

1. AUTOCOMMIT is set to FIN. No COMMIT is issued unless specifically coded. As is required for this AUTOCOMMIT setting, AUTODISCONNECT and AUTOCLOSE are also set to FIN.

2. Table XYZ is locked for this program’s exclusive use. This lock will be terminated at a COMMIT or ROLLBACK point.

3. The procedure checks &RETCODE. If it is not zero, the SQLCODE is displayed to show that the lock was not completed; the procedure issues a ROLLBACK and terminates. This &RETCODE test is executed after every SQL statement passed to DB2.

4. Three rows are inserted into XYZ.

5. The program issues an explicit COMMIT making the inserts to XYZ permanent and releasing the exclusive lock.

6. SET AUTOCOMMIT is reissued to restore automatic command-level COMMITs.

11 Environmental Commands

Topics:

- SQL ? Query
- DBSPACE
- DEFDATE
- EXPLAIN
- ISOLATION
- OPTIMIZATION
- PASSRECS
- SQLJOIN OUTER
- DB2 Only
- MODIFY Only
- Interface Dialogue Manager Variables

Interface environmental commands can change certain parameters that govern your FOCUS session. These parameters control or identify the Change Verify Protocol, DB2 Call Attachment Facility defaults, DBSPACE defaults, the Fastload option, the DB2 CURRENT SQLID, Interface Optimization, and Isolation Levels. Environmental commands that control connection to the RDBMS, like SET AUTODISCONNECT, are discussed in Chapter 10, *Thread Control Commands*.

You can display all the current parameter settings with the Interface SQL ? query described in *SQL ? Query* on page 11-3.

There are two forms of acceptable syntax for Interface commands. (Commands that you issue from MODIFY procedures, like LOADONLY, ERRORRUN, and AUTOCOMMIT ON CRTFORM, use a slightly different form of syntax described in *MODIFY Only* on page 11-17.) Use the first form of syntax to invoke the Direct SQL Passthru facility (see Chapter 9, *Direct SQL Passthru*). The first form is

```
SQL [target_db] SET command  value
```

where:

target_db
: Indicates the target RDBMS. Valid values are DB2 or SQL/DS. Omit if you previously issued the SET SQLENGINE command.

command
: Is an environmental command.

value
: Is an acceptable value for the environmental command.

Use the second form of syntax if you do not want to invoke the Direct SQL Passthru facility. This alternative syntax includes TSO, MVS, or CMS environmental qualifiers; the SQL keyword is required

```
environment SQL SET command   value
```

where:

environment
: Is one of the following: TSO, MVS, or CMS.

Although the commands are prefixed with environmental qualifiers TSO, MVS, or CMS, the Interface processes them, not the operating system.

Note: You can use the TSO and MVS environmental qualifiers interchangeably in any MVS environment (TSO, batch processing, MSO).

For the remainder of this chapter, assume Direct SQL Passthru is in effect. The syntax examples show only the Passthru form of each command. If you are not using Direct SQL Passthru, see the preceding syntax instructions.

SQL ? Query

The Interface SQL query command displays Interface defaults and current settings for DB2 (CAF and non-CAF versions) and SQL/DS.

To check your defaults, issue the query from the FOCUS command level. Include a TSO SQL, MVS SQL, or CMS SQL prefix if Direct SQL Passthru is not in effect. The syntax is

```
SQL [target_db] ?
```

where:

target_db
: Indicates the target RDBMS. Valid values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command,

The following is an example:

```
> sql db2 ?
(FOC1440) CURRENT SQL INTERFACE SETTINGS ARE :
(FOC1442) CALL ATTACH FACILITY IS             -  : ON
(FOC1447) SSID FOR CALL ATTACH IS             -  : DSN
(FOC1448) ACTIVE PLAN FOR CALL ATTACH IS      -  :
(FOC1459) USER SET PLAN FOR CALL ATTACH IS    -  :
(FOC1460) INSTALLATION DEFAULT PLAN     IS    -  : P7009701
(FOC1503) SQL STATIC OPTION IS                -  : OFF
(FOC1444) AUTOCLOSE OPTION IS                 -  : ON FIN
(FOC1496) AUTODISCONNECT OPTION IS            -  : ON FIN
(FOC1499) AUTOCOMMIT OPTION IS                -  : ON COMMAND
(FOC1446) DEFAULT DBSPACE IS                  -  :
(FOC1449) CURRENT SQLID IS                    -  : SYSTEM DEFAULT
(FOC1424) ISOLATION LEVEL FOR DB2 TABLE INTERFACE IS  :
(FOC1441) WRITE FUNCTIONALITY IS              -  : ON
(FOC1445) OPTIMIZATION OPTION IS              -  : ON
(FOC1484) SQL ERROR MESSAGE TYPE IS           -  : FOCUS
(FOC1497) SQL EXPLAIN OPTION IS               -  : OFF
(FOC1552) INTERFACE DEFAULT DATE TYPE         -  : NEW
>  >
```

DBSPACE

Within a FOCUS session, you can designate a default storage space for tables you create with the FOCUS CREATE FILE or HOLD FORMAT SQL commands. For the duration of the session, the RDBMS places such tables in the SQL/DS DBSPACE or DB2 database you identify in the SET DBSPACE command.

Issue the SET DBSPACE command from the FOCUS command level

```
SQL [target_db] SET DBSPACE storage
```

where:

target_db
: Indicates the target RDBMS. Valid values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

storage
: For DB2, is databasename.tablespacename or DATABASE databasename. In DB2, the RDBMS default value is DSNDB04, a public database. (DB2 automatically generates a tablespace in DSNDB04.)

  For SQL/DS, is owner.dbspace. In SQL/DS, the RDBMS default value is the SQL/DS authorization ID’s private DBSPACE.

The Interface may have been installed with a default, site-specific, DBSPACE specification. Use the SQL ? query command to display the setting.

DEFDATE

Use the SET DEFDATE command to change the value of the default date FOCUS uses for the RDBMS DATE datatype. See *Default DATE Considerations* in Appendix A, *Additional Topics*, for a complete discussion.

From the FOCUS command line, issue

```
SQL [target_db] SET DEFDATE {NEW|OLD}
```

where:

target_db
: Indicates the target RDBMS. Valid values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

NEW
: Supplies '1900-12-31' as the base date for RDBMS columns with DATE datatypes. NEW is the default starting in FOCUS 6.8.

`OLD`

Supplies '1901-01-01' as the base date for RDBMS columns with DATE datatypes. OLD was the default in versions of the Interface prior to 6.8.

If you use the FOCUS MODIFY facility to maintain RDBMS tables containing DATE columns, the change in the default date from prior releases may impact existing applications. See *Effects on Existing Applications* in Appendix A, *Additional Topics*.

EXPLAIN

You can instruct the Interface to execute an RDBMS EXPLAIN command for the SQL SELECT statements issued by a FOCUS request before it actually issues the FOCUS request. At the FOCUS command level, enter

```
SQL [target_db] SET EXPLAIN {OFF|ON [n]}
```

where:

`target_db`

Indicates the target RDBMS. Valid values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

`OFF`

Is the default. The Interface proceeds as usual.

`ON`

Instructs the Interface to issue an RDBMS EXPLAIN command for the SQL SELECT statements issued by a FOCUS request, then issue the request itself. (Do not confuse this with the Interface EXP facilities.)

`n`

Is the query number to use in the RDBMS EXPLAIN tables. The default value is 1.

To use the ON setting, you must satisfy all RDBMS requirements for executing an EXPLAIN, such as having EXPLAIN tables. The ON setting applies to TABLE and SQL Passthru SELECT requests only; it is ignored by the MODIFY facility and non-SELECT SQL Passthru requests.

ISOLATION

Each RDBMS protects data being read by one user from changes (INSERT, UPDATE, or DELETE) made by others; the Isolation Level setting governs the duration of the protection. That is, the Isolation Level determines when shared locks on rows or data pages are released, so that those rows or pages become available for updates by other users. You can dynamically set the Isolation Level within the FOCUS session using the SET ISOLATION command.

You can change the isolation level by issuing the SET ISOLATION command in a MODIFY procedure or at the FOCUS command level. (For MAINTAIN, you must issue the SET ISOLATION command at the FOCUS command level prior to invoking the MAINTAIN procedure.) The setting remains in effect for the FOCUS session or until you reset it.

From the FOCUS command level, issue

```
SQL [target_db] SET ISOLATION level
```

where:

target_db
: Indicates the target RDBMS. Valid values are DB2 or SQL/DS. Omit if you previously issued the SET SQLENGINE command.

level
: Indicates the isolation level. Valid values are:

CS	Is Cursor Stability, the default. Releases shared locks as the cursor moves on in the table. Use for read-only requests.
RR	Is Repeatable Read. Use for MODIFY and MAINTAIN read/write routines. Locks the retrieved data until it is released by an SQL COMMIT WORK or SQL ROLLBACK WORK.
UR	Is Uncommitted Read. Available in DB2 only, Version 4 and higher. It provides read-only access to records even if they are locked; however, these records may not yet be committed to the database.
RS	Is Read Stability. Available in DB2 only, Version 5.1 and higher. For more information, see the *DB2 Command and Utility Reference.*
NC	Available in DB2 only, Version 5.1 and higher. For more information, see the *DB2 Command and Utility Reference.*
blank	A blank value resets the level to the Interface default.

Note:

- Omit the target RDBMS qualifier to issue the command in MODIFY procedures or if you previously issued the SET SQLENGINE command.
- The Interface does not validate the isolation level values. If you issue the SET ISOLATION command with a level not supported for the version of the RDBMS you are using, the RDBMS will return an SQL code of -104, signifying an SQL syntax error.
- For SQL/DS, the isolation level you set applies to locks held on tables in public dbspaces. (SQL/DS always locks private dbspaces at the dbspace level.)
- For DB2:
    - The SET ISOLATION command is enabled only for SELECT requests created as a result of FOCUS TABLE requests.

- This setting cannot override the required isolation level of RR for MODIFY and MAINTAIN requests.

To display the isolation level setting, issue the SQL ? query command.

OPTIMIZATION

Depending on the OPTIMIZATION setting, the Interface may generate SQL SELECT statements that allow the RDBMS to perform operations (such as join and aggregation) and return the data to the Interface for FOCUS report generation (see Chapter 7, *The Interface Optimizer*).

To invoke Interface optimization, enter the following at the FOCUS command level

```
SQL [target_db] SET {OPTIMIZATION|SQLJOIN} optsetting
```

where:

target_db
: Indicates the target RDBMS. Valid values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

SQLJOIN
: Is an alias for OPTIMIZATION.

optsetting
: Indicates whether the Interface should pass sort, join, and aggregation operations to the RDBMS for processing. Valid values are as follows:

OFF	Instructs the Interface to create SQL statements for simple data retrieval from each table. FOCUS processes the returned sets of data in your address space or virtual machine to produce the report.
ON	Instructs the Interface to create SQL statements that take advantage of RDBMS join, sort, and aggregation capabilities. Is compatible with previous releases in regard to the multiplicative effect: misjoined unique segments and multiplied lines in PRINT and LIST based report requests do not disable optimization. Other cases of the multiplicative effect invoke Interface-managed native join logic. See Chapter 7, *The Interface Optimizer*, for more information. ON is the default.
FOCUS	Passes join logic to the RDBMS only when the results will be the same as from a FOCUS-managed request. Misjoined unique segments, the multiplicative effect, and multiplied lines in PRINT and LIST based requests invoke Interface-managed native join logic. See Chapter 7, *The Interface Optimizer*, for information.
SQL	Passes join logic to the RDBMS in all possible cases. The multiplicative effect does not disable optimization, even in cases involving aggregation (SUM, COUNT). Join logic is not passed to the RDBMS for tables residing on multiple subsystems and for tables residing on multiple DBMS platforms.

PASSRECS

You can use the SET PASSRECS command to display the number of rows affected by a successfully executed Direct SQL Passthru UPDATE or DELETE command. The syntax is

```
SQL [target_db] SET PASSRECS {OFF|ON}
```

where:

target_db
: Indicates the target RDBMS. Valid values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

`OFF`
: Is the default. As in previous releases, the Interface provides no information as to the number of records affected by a successfully executed Direct SQL Passthru UPDATE or DELETE command.

`ON`
: Provides the following FOCUS message after the successful execution of a Direct SQL Passthru UPDATE or DELETE command:

```
(FOC1364) ROWS AFFECTED BY PASSTHRU COMMAND: #/operation
```

For example, a DELETE command that executes successfully and affects 20 rows generates the following message:

```
(FOC1364) ROWS AFFECTED BY PASSTHRU COMMAND: 20/DELETE
```

In addition to this message, the Interface updates the FOCUS system variable &RECORDS with the number of rows affected. You can access this variable via Dialogue Manager and display it with the ? STAT query.

Note:

- Although you can use native SQL syntax (see Appendix B, *Native SQL*) to issue the SET PASSRECS command, if you use it to issue subsequent UPDATE or DELETE commands, the Interface will not issue the FOC1364 message or update the &RECORDS system variable. You must use Direct SQL Passthru syntax to issue UPDATE or DELETE commands in order to invoke the SET PASSRECS command.
- Since, by definition, the successful execution of an INSERT command always affects one record, INSERT does not generate the FOC1364 message.
- The FOC1364 message is for informational purposes only and does not affect the &FOCERRNUM setting.

SQLJOIN OUTER

With the SET SQLJOIN OUTER command you can control when the Interface optimizes outer joins without affecting the optimization of other operations. This parameter provides backward compatibility with prior releases of the Interface and enables you to fine-tune your applications.

When join optimization is in effect, the Interface generates one SQL SELECT statement that includes every table involved in the join. The RDBMS can then process the join. When join optimization is disabled, the Interface generates a separate SQL SELECT statement for each table, and FOCUS processes the join.

The syntax is

```
SQL target_db SET SQLJOIN OUTER {ON|OFF}
```

where:

target_db
: Indicates the target RDBMS. Valid values are DB2 or SQL/DS. Omit if you issued the SET SQLENGINE command.

`ON`
: Enables outer join optimization. ON is the default value.

`OFF`
: Disables outer join optimization.

Note:

- The SQLJOIN OUTER setting is available only when optimization is enabled (that is, OPTIMIZATION is not set to OFF).
- The SQLJOIN OUTER setting is ignored when SET ALL = OFF.

The following table describes how different combinations of OPTIMIZATION and SQLJOIN OUTER settings affect Interface behavior. It assumes that SET ALL = ON:

Settings		**Results**	
OPTIMIZATION	**SQLJOIN OUTER**	**Outer Join Optimized?**	**Other Optimization Features**
ON	ON	Yes	Enabled
ON	OFF	No	Enabled
OFF	N/A	No	Disabled
SQL	ON	Yes, in all possible cases	Enabled
SQL	OFF	No	Enabled
FOCUS	ON	Yes if results are equivalent to FOCUS managed request	Enabled
FOCUS	OFF	No	Enabled

If SQLJOIN OUTER is set to OFF, the following message displays when you issue the SQL ? query command:

```
(FOC1420) OPTIMIZATION OF ALL=ON AS LEFT JOIN -  : OFF
```

DB2 Only

The following Interface environmental commands apply only to DB2: SET BINDOPTIONS, SET CURRENT DEGREE, SET ERRORTYPE, SET CURRENT SQLID, SET IXSPACE, SET OPTIFTHENELSE, SET OWNERID, SET SSID, and SET PLAN. (SET AUTOCLOSE also applies to DB2 only and is discussed in Chapter 10, *Thread Control Commands*.)

BINDOPTIONS

Starting with FOCUS Version 7.0, you can override the default BIND string that the Interface creates when you compile a static MODIFY procedure (with SET STATIC ON) or TABLE request (with SET STATIC OFF or ON).

The syntax is

```
SQL [DB2] SET BINDOPTIONS [bind_spec]
```

where:

bind_spec

Is the portion of the BIND command that contains the BIND keywords and parameters (*not* including the BIND keyword itself). These BIND keywords and parameters must conform to rules governing the BIND PLAN and BIND PACKAGE commands described in the IBM *DB2 Command and Utility Reference.*

Note: Do not include the word BIND as part of the bind_spec.

To reset the default options, issue the command with no bind_spec:

```
SQL DB2 SET BINDOPTIONS
```

Note: Omit the DB2 target RDBMS qualifier if you previously issued the SET SQLENGINE command for DB2.

Use the long form of Direct SQL Passthru syntax for commands that exceed one line:

```
SQL DB2
SET BINDOPTIONS bind_spec
END
```

The Interface default BIND string has the form:

```
BIND PLAN (focexec_name) MEM(focexec_name) ACTION(REPLACE) ISOLATION(CS)
```

You can determine the current setting for BINDOPTIONS with the Interface SQL DB2 ? query command. If the current setting is the default, BINDOPTIONS does not display in the SQL DB2 ? output.

Note: If the bind string specifies the MEM keyword, the Interface ignores it and supplies the MEM keyword with a parameter value equal to the FOCEXEC name. If the bind string omits any of the other three default keywords, PLAN/PACKAGE, ACTION, or ISOLATION, the Interface adds them automatically with their corresponding default values. To bind a package, you must specify the keyword PACKAGE, as PLAN is the default.

CURRENT DEGREE

DB2 Version 3 supports parallel query I/O and Version 4 supports parallel query CPU to improve response. You can bind static SQL requests with the DEGREE(ANY) parameter to take advantage of parallel processing. For dynamic SQL requests, the Interface supports parallel processing if you issue the SET CURRENT DEGREE command prior to the request.

The syntax is

```
SQL [DB2] SET CURRENT DEGREE {'1'|'ANY'}
```

where:

1

Is the default; does not invoke parallel processing.

`ANY`

Invokes parallel processing for dynamic requests. If the thread to DB2 is closed during the session, the value resets to ‘1’.

Note:

- Omit the DB2 target RDBMS qualifier if you previously issued the SET SQLENGINE command for DB2.
- Closing a DB2 thread restores the setting to ‘1’, the default. Therefore, if you set AUTOCLOSE or AUTODISCONNECT to anything other than ON FIN (the Interface default), you must reissue the SET CURRENT DEGREE command prior to each request in which you want it to apply.

CURRENT SQLID

The DB2 RDBMS recognizes two types of ID, the primary authorization ID and one or more optional secondary authorization IDs; it also recognizes the CURRENT SQLID setting.

An ID known as a primary authorization ID identifies an interactive user or batch program accessing a DB2 subsystem. A security system such as RACF normally provides the ID to DB2. During the process of connecting to DB2, the primary authorization ID may be associated with one or more secondary authorization IDs (usually RACF groups). The site controls whether it uses secondary authorization IDs.

The DB2 database administrator may grant privileges to a secondary authorization ID that are not granted to the primary ID. Thus, secondary authorization IDs provide the means for granting the same privileges to a group of users. (The DBA associates individual primary IDs with the same secondary ID and then grants the privileges to the secondary ID.)

The DB2 CURRENT SQLID can be the primary authorization ID or any associated secondary authorization ID. At the beginning of the FOCUS session, the CURRENT SQLID is the primary authorization ID. You can reset it using the following Interface command

```
SQL [DB2] SET CURRENT SQLID = 'sqlid'
```

where:

`sqlid`

Is the desired primary or secondary authorization ID, enclosed in single quotation marks. All DB2 security rules are respected.

Note: Omit the DB2 target RDBMS qualifier if you previously issued the SET SQLENGINE command for DB2.

Unless you issue the SET OWNERID command, the CURRENT SQLID is the default owner ID for DB2 objects (such as tables or indexes) created with dynamic SQL statements. (For example, the FOCUS CREATE FILE command issues dynamic SQL statements.) The CURRENT SQLID is also the sole authorization ID for GRANT and REVOKE statements. It must be assigned all the privileges needed to create objects as well as GRANT and REVOKE privileges. If you do not issue the SET OWNERID command, the CURRENT SQLID is assumed to be the owner for unqualified table names.

Other types of requests, such as FOCUS TABLE (SQL SELECT) and MODIFY (SQL SELECT, INSERT, UPDATE, or DELETE) requests, automatically search for the necessary privileges using the combined privileges of the primary authorization ID and all of its associated secondary authorization IDs, regardless of the DB2 CURRENT SQLID setting. The CURRENT SQLID setting stays in effect until the communication thread to DB2 is disconnected, when it reverts to the primary authorization ID.

ERRORTYPE

With SET ERRORTYPE, you can instruct the DB2 Interface to return native DB2 error messages, as well as FOCUS error messages, for those error conditions that are reported by the DBMS. This feature can be enabled both as an installation option and as a run time SET parameter.

You can override the installation default setting at run time. The syntax is

```
SQL [DB2] SET ERRORTYPE {FOCUS|DBMS}
```

where:

`FOCUS`
: Generates only FOCUS error messages.

`DBMS`
: Produces native DBMS error messages as well as FOCUS error messages.

Note:

- Omit the DB2 target RDBMS qualifier if you previously issued the SET SQLENGINE command for DB2.
- The SET option overrides the installation default.
- This command is not available for SQL/DS.

IXSPACE

Starting with FOCUS Version 7.0, you can override the default parameters for the DB2 index space implicitly created by FOCUS CREATE FILE and HOLD FORMAT DB2. The syntax is

```
SQL [DB2] SET IXSPACE [index_spec]
```

where:

index_spec

Is the portion (up to 94 bytes) of the SQL CREATE INDEX statement beginning with the USING-BLOCK (as specified in the *IBM DB2 SQL Reference* CREATE INDEX syntax diagram).

To reset to the DB2 default index space parameters, issue the SET IXSPACE command with no operands.

Note: Omit the DB2 target RDBMS qualifier if you previously issued the SET SQLENGINE command for DB2.

Use the long form of Direct SQL Passthru syntax for commands that exceed one line:

```
SQL DB2
SET IXSPACE index_spec
END
```

For example, to specify the USING-BLOCK, FREE-BLOCK, and CLUSTER portions of the CREATE INDEX statement, enter the following:

```
SQL DB2
SET IXSPACE USING STOGROUP SYSDEFLT PRIQTY 100
SECQTY 20 FREEPAGE 16 PCTFREE 5 CLUSTER
END
```

You can use the SQL ? query command to determine the current IXSPACE setting. If the current setting is the default, IXSPACE does not display in the SQL DB2 ? output.

OPTIFTHENELSE

When you issue the SET OPTIFTHENELSE command, the Interface attempts to deliver the construct of FOCUS IF-THEN-ELSE DEFINE fields to DB2 as an expression. The DEFINE field must be an object of a selection test or an aggregation request. The DEFINE definition may be specified in the TABLE request or in the Master File.

```
SQL {DB2} SET OPTIFTHENELSE {ON|OFF}
```

where

`ON`

Enables IF-THEN-ELSE optimization.

`OFF`

Disables IF-THEN-ELSE optimization and is the default

Note: Omit the DB2 target RDBMS qualifier to issue the command if you previously issued the SET SQLENGINE command for DB2.

OWNERID

You can issue the SET OWNERID command to designate a creator name for all unqualified table names. The Interface will then use the owner ID as the creator name whenever the Access File does not include a creator value as part of the table name. Direct SQL Passthru requests are not affected by this setting. The syntax is

```
SQL {DB2} SET OWNERID ownerid_value
```

where:

`ownerid_value`

Is the owner ID to assign to all unqualified table names.

Note: Omit the DB2 target RDBMS qualifier to issue the command if you previously issued the SET SQLENGINE command for DB2.

The Interface uses the owner ID whenever there is ambiguity about the creator name. For example, the Interface uses the owner ID as the creator for any table you create using:

- The CREATE FILE command when the Access File does not specify a creator.
- HOLD FORMAT DB2 when the name of the extract file does not include a period.

If you do not have sufficient rights to create tables using the owner ID you set, an SQL error results and the table is not created.

When this setting is in effect, the following line is added to the output of the SQL DB2 ? query:

```
(FOC1520)  SQL CURRENT OWNER ID IS ownerid
```

PLAN

In addition to the DB2 subsystem-ID, you must identify your DB2 application plan before you execute any requests. The plan is a result of the Interface installation process or, possibly, the result of compiling a procedure that uses static SQL. See Chapter 13, *Static SQL*, for a discussion of static SQL procedures.

In a CAF environment, issue the SET PLAN command from the FOCUS command level

```
SQL [DB2] SET PLAN planname
```

where:

`planname`

Is the name of your application plan.

Note:

- Omit the DB2 target RDBMS qualifier if you previously issued the SET SQLENGINE command for DB2.
- The default Isolation Level for the DB2 Interface is CS. At installation time, your site may have chosen RR as the Isolation Level. Your DB2 database administrator can provide you with the Isolation Level for a given Interface application plan. Refer to *ISOLATION* on page 11-5 for more information.

  In DB2 Version 4, there is an additional Isolation Level called Uncommitted Read (UR). It provides read-only access to records even if they are locked; however, these records may not yet be committed to the database. For more information, see the *DB2 Command and Utility Reference*.
- Sometimes, a site binds two different plans during the installation process; each specifies a different Isolation Level to control RDBMS locking. You can use the SET PLAN command to, in effect, change your Isolation Level within a FOCUS session. A level of repeatable read (RR) is required for read/write MODIFY and MAINTAIN applications including those that invoke the Change Verify Protocol (CVP); cursor stability (CS) is recommended for TABLE and read-only MODIFY or MAINTAIN operations. Refer to *ISOLATION* on page 11-5 for more information.

  This technique does not apply to non-CAF versions of the Interface; they require a different CLIST or JCL procedure to invoke different versions of the Interface.

You can reset the plan name at any time, regardless of the AUTOCLOSE setting. If you change the plan name, the next native SQL command or FOCUS request uses the new plan by closing and re-opening the thread.

The Interface may have been installed with a default, site-specific, PLAN setting. Use the SQL ? query command to display this setting.

SSID

If the CAF option for the Interface is installed at your site, you must indicate which DB2 system you intend to use. The name for the DB2 system may differ from the default, or your site may have multiple copies of DB2. To specify the DB2 subsystem-ID (SSID), issue the SET SSID command before executing native SQL commands or FOCUS requests.

Issue the following from the FOCUS command level or include it in a PROFILE FOCEXEC

```
SQL [DB2] SET SSID ssid
```

where:

ssid
: Is the DB2 subsystem ID; the default is DSN, unless your site changed the default at installation time.

Note:

- Omit the DB2 target RDBMS qualifier if you previously issued the SET SQLENGINE command for DB2.
- Non-CAF versions of the Interface usually supply the subsystem-ID by specifying the DSN SYSTEM parameter and the RUN subcommand in the TSO CLIST that invokes FOCUS.
- Under MSO, there can be only one SSID per address space; it is identified during installation and may not be changed.

You can reset the SSID at any time, regardless of the AUTOCLOSE setting. If you change the SSID setting, the next native SQL command or FOCUS request accesses the new DB2 subsystem.

The Interface may have been installed with a default, site-specific, SSID setting. Use the SQL ? query command to display this setting.

MODIFY Only

The following commands are available only within FOCUS MODIFY procedures: SET LOADONLY, SET AUTOCOMMIT ON CRTFORM, and SET ERRORRUN. The SET ISOLATION command, explained in *ISOLATION* on page 11-5 may also be issued from MODIFY. See Chapter 12, *Maintaining Tables With FOCUS*, for a discussion of MODIFY.

When issuing commands from within MODIFY procedures, omit the target_db or the environmental prefix.

LOADONLY

The Fastload facility increases the speed of loading data into tables. Use the LOADONLY command exclusively in those MODIFY procedures that insert rows. The syntax is:

```
SQL SET LOADONLY
```

You can use the Fastload feature only with ON NOMATCH INCLUDE operations; other MODIFY operations generate an error message if SQL SET LOADONLY is invoked.

In MODIFY processing without Fastload, the Interface uses an SQL SELECT statement to test the existence of a single row. By examining the SQL return code, the Interface determines whether the row exists and directs MODIFY processing to the appropriate ON MATCH or ON NOMATCH logic.

The Fastload option eliminates this SELECT operation. It loads rows into the table without first evaluating their existence. The RDBMS ensures the uniqueness of stored rows with unique indexes.

Note: In the SQL/DS Interface, LOADONLY uses the SQL/DS blocked-insert statement, PUT. Any non-zero return code from a PUT operation (such as a unique index violation) may result in failure to input more than one row. SQL/DS Interface users using LOADONLY should test FOCERROR after every INCLUDE. If a non-zero code is returned, in all cases, applications should ROLLBACK and terminate the MODIFY procedure.

AUTOCOMMIT ON CRTFORM

The AUTOCOMMIT ON CRTFORM facility provides an automated Change Verify Protocol (CVP) as an alternative to the RDBMS standard method of locking for concurrency and integrity. The Change Verify Protocol consists of a series of steps that manage the integrity and concurrent update/retrieval activity of the database by committing open transactions prior to displaying each CRTFORM.

AUTOCOMMIT ON CRTFORM does not apply to MAINTAIN.

CVP lacks the unit-of-work capabilities of the RDBMS protocol, but CVP never retains locks on displayed records. The elimination of locks increases rates of transaction operations and improves terminal response times for interactive applications. All open transactions are automatically committed prior to using CVP for SELECT and UPDATE.

The Change Verify Protocol initiates these steps for each CRTFORM display:

1. The Interface releases all locks held on the record. Before the record displays on the terminal, the Interface issues an SQL COMMIT WORK statement to release any existing RDBMS locks.
2. The Interface retrieves the record and displays it on the terminal. The user enters a database update that results in an UPDATE, DELETE, or INCLUDE action.
3. The Interface retrieves the record again. It compares the original and current values of the record to determine if another transaction changed the record in the interim. If the original (displayed) record has been modified in the time between the original retrieval and the update request, the update is rejected. If the original (displayed) record remains unchanged, the update is processed.

With the RDBMS standard method, SQL COMMIT WORK statements specified in applications define transactions or logical units of work (LUW). The RDBMS uses a locking mechanism to protect the LUW from interference by other concurrent transactions. The locking mechanism allocates and isolates database resources for the LUW.

The disadvantage to the RDBMS method is that terminal I/O in the LUW locks data for the indeterminate amount of time it takes the user to react to the terminal display. The locks are retained until the user completes the transactions successfully or until the RDBMS issues a ROLLBACK WORK command.

You can issue the AUTOCOMMIT ON CRTFORM command only from within a MODIFY CRTFORM procedure. Include it in CASE AT START, not in the TOP case. You may not switch AUTOCOMMIT modes within the MODIFY procedure.

The AUTOCOMMIT ON CRTFORM facility only works on single-record transactions. (For example, MODIFY MATCH and NEXT commands retrieve single records.) It is not designed for set processing with FOCUS multiple-record operations (REPEAT and HOLD, for example). For multiple-record processing, CVP applies only to the last record.

The syntax is

```
SQL SET AUTOCOMMIT {OFF|ON CRTFORM}
```

where:

`OFF`

Is the default; it retains the native RDBMS locking protocol.

`ON CRTFORM`

Invokes the Change Verify Protocol.

Note:

- A MODIFY procedure that invokes the Change Verify Protocol can test the value of the FOCURRENT variable to determine whether or not there is a conflict with another transaction. See *The FOCURRENT Variable* in Chapter 12, *Maintaining Tables With FOCUS*, for more information.
- To avoid data inconsistencies with the AUTOCOMMIT feature, you must set your SQL Isolation Level to repeatable read (RR), the required Isolation Level for all read/write MODIFY applications.

  For SQL/DS, change the level by issuing the SET ISOLATION command from the FOCUS command level or from within MODIFY. For DB2, you may need to use another DB2 application plan that is bound with the RR level. Issue the SET PLAN command to change DB2 application plans.

ERRORRUN

With SET ERRORRUN ON, MODIFY processing continues even when a serious error occurs, allowing an application to handle its own errors in the event that an RDBMS error is part of the normal application flow. Code this command explicitly within the MODIFY procedure, preferably in CASE AT START, where it is executed once.

The syntax is

```
CASE AT START

SQL SET ERRORRUN {OFF|ON}
.
.
.
ENDCASE
```

where:

`OFF`

Causes MODIFY processing to stop when a fatal error is detected (for example, if the table name is not found). OFF is the default value.

`ON`

Allows MODIFY processing to continue despite fatal errors. Test the value of FOCERROR to determine appropriate action after an RDBMS call.

When SET ERRORRUN is ON, the MODIFY procedure reports the error but continues execution. The MODIFY code can then test the value of FOCERROR to determine the cause of the error and take appropriate action. Be careful in evaluating the contents of FOCERROR, as failure to respond to a negative SQLCODE can cause unpredictable errors in subsequent RDBMS or MODIFY processing.

SET ERRORRUN returns to its default setting at the end of the MODIFY procedure.

Interface Dialogue Manager Variables

All Interface environmental settings are available for display and query as Dialogue Manager variables. The following sections list the variable names for each RDBMS, and the lines displaying their settings in the SQL ? query.

Note: If the length of the current setting value is greater than 12, the Dialogue Manager variable makes only the first 12 characters available.

Dialogue Manager Variables for the DB2 Interface

The following variables are available for DB2:

```
&SQLCAF       - (FOC1442) CALL ATTACH FACILITY IS
&SQLSSID      - (FOC1447) SSID FOR CALL ATTACH IS
&SQLPLANA     - (FOC1448) ACTIVE PLAN FOR CALL ATTACH IS
&SQLPLANU     - (FOC1459) USER SET PLAN FOR CALL ATTACH IS
&SQLPLANI     - (FOC1460) INSTALLATION DEFAULT PLAN IS
&SQLSTATIC    - (FOC1503) SQL STATIC OPTION IS
&SQLAUTOCLS   - (FOC1444) AUTOCLOSE OPTION IS
&SQLAUTODIS   - (FOC1496) AUTODISCONNECT OPTION IS
&SQLAUTOCOM   - (FOC1499) AUTOCOMMIT OPTION IS
&SQLDBSPACE   - (FOC1446) DEFAULT DBSPACE IS
&SQLID        - (FOC1449) CURRENT SQLID IS
&SQLISOLATION - (FOC1424) ISOLATION LEVEL FOR DB2 TABLE INTERFACE IS
&SQLWRITE     - (FOC1441) WRITE FUNCTIONALITY IS
&SQLOPT       - (FOC1445) OPTIMIZATION OPTION IS
&SQLEMSG      - (FOC1484) SQL ERROR MESSAGE TYPE IS
&SQLEXPL      - (FOC1497) SQL EXPLAIN OPTION IS
&SQLQUERYNO   - (FOC1551) QUERY NUMBER TO BE USED FOR EXPLAIN
&SQLDEFDATE   - (FOC1552) INTERFACE DEFAULT DATE TYPE

&SQLRELEASE   - Current release of DB2 in format Dxxx where
                    xxx is release number (for example, 510 for
                    DB2 Version 5 Release 1).

&SQLVVRRM     - Return from CONNECT in the form DSNvvrrM.
                 vv is the Version of DB2, rr is the Release, M is the
                 Modification level. For more information, consult IBM’s
                 DB2 SQL Reference.
```

Dialogue Manager Variables for the SQL/DS Interface

The following variables are available for SQL/DS:

```
&SQLSTATIC   - (FOC1503) SQL STATIC OPTION IS
&SQLAUTODIS  - (FOC1496) AUTODISCONNECT OPTION IS
&SQLAUTOCOM  - (FOC1499) AUTOCOMMIT OPTION IS
&SQLDBSPACE  - (FOC1446) DEFAULT DBSPACE IS
&SQLWRITE    - (FOC1441) WRITE FUNCTIONALITY IS
&SQLOPT      - (FOC1445) OPTIMIZATION OPTION IS
&SQLEMSG     - (FOC1484) SQL ERROR MESSAGE TYPE IS
&SQLEXPL     - (FOC1497) SQL EXPLAIN OPTION IS
&SQLQUERYNO  - (FOC1551) QUERY NUMBER TO BE USED FOR EXPLAIN
&SQLDEFDATE  - (FOC1552) INTERFACE DEFAULT DATE TYPE

&SQLVVRRM    - Return from CONNECT in the form ARIvvrrM.
                vv is the Version of SQL/DS, rr is the Release, M is the
                Modification level. For more information, consult IBM’s
                SQL/DS SQL Reference.
```

12 Maintaining Tables With FOCUS

Topics:

- Types of Relational Transaction Processing
- The Role of the Primary Key
- Index Considerations
- Modifying Data
- Referential Integrity
- The MODIFY COMBINE Facility
- The LOOKUP Function
- The FIND Function
- Isolation Levels
- Issuing SQL Commands in MODIFY
- The Change Verify Protocol: AUTOCOMMIT ON CRTFORM
- Loading Tables Faster With MODIFY: The Fastload Facility

This chapter describes how to use the Interface to maintain RDBMS tables. In particular, it identifies aspects of the FOCUS file maintenance facilities, MODIFY and MAINTAIN, that are unique for the RDBMS environment. MODIFY and MAINTAIN requests read, add, update, and delete rows in tables. You can modify single tables, sets of tables defined in a multi-table Master File, or unrelated sets of tables.

MAINTAIN provides a graphical user interface and event-driven processing. In a MAINTAIN procedure, temporary storage areas called stacks collect data, transaction values, and temporary field values. You can use the MAINTAIN Window Painter facility to design Winforms—windows that display stack values, collect transaction values, and invoke triggers. A trigger implements event-driven processing by associating an action (such as performing a specific case in the MAINTAIN procedure) with an event (such as pressing a particular PF key). MAINTAIN also provides set-based processing through enhanced NEXT, UPDATE, DELETE, and INCLUDE commands.

This chapter describes differences between the MODIFY and MAINTAIN facilities when these differences affect Interface processing.

The *FOCUS for IBM Mainframe User's Manual* contains a detailed discussion of file maintenance with the FOCUS MODIFY facility. Read the MODIFY chapter carefully before developing MODIFY procedures to use with RDBMS tables. The *FOCUS for IBM Mainframe MAINTAIN User's Manual* contains a detailed description of the FOCUS MAINTAIN facility. Read it carefully before developing MAINTAIN procedures to use with RDBMS tables.

Note: You can maintain up to 64 tables in a single MODIFY or MAINTAIN procedure. The limit for a MODIFY COMBINE or a MAINTAIN procedure is 16 Master Files; however, each Master File can describe more than one table, for a total of 64 segments per procedure minus one for the artificial root segment created by the COMBINE command. In addition, a MAINTAIN procedure can call other MAINTAIN procedures that reference additional tables.

Prerequisites for running MODIFY and MAINTAIN requests include:

- The Write Interface. The Write component of the Interface must be installed and operational.
- Proper RDBMS authorization to perform maintenance operations.

  You can update some RDBMS views in accordance with database rules. In general, the RDBMS permits updates to views that are subsets of columns, rows, or both; it does not permit updates to views that perform joins or involve aggregation. For rules regarding the maintenance of RDBMS views, see the SQL Reference Manual for the appropriate RDBMS.
- A WRITE keyword value of YES in the table's Access File (for INCLUDE, UPDATE, and DELETE operations). Consult *WRITE* in Chapter 4, *Describing Tables to FOCUS*, for information about this keyword.
- Existing tables to modify. If you intend to load new tables with MODIFY or MAINTAIN, you must generate them first. Do so with the FOCUS CREATE FILE command, discussed in Chapter 6, *Automated Procedures*, or with the SQL CREATE TABLE command through the Direct SQL Passthru facility, discussed in Chapter 9, *Direct SQL Passthru*.

  You can maintain views as long as they are updatable, as defined in the IBM *DB2 SQL Reference*.

The examples in this chapter refer to the EMPINFO, COURSE, PAYINFO, ECOURSE, and EMPPAY Master and Access Files. Appendix C, *File Descriptions and Tables*, contains a complete listing of Master and Access Files.

Types of Relational Transaction Processing

You can process incoming transactions by comparing (or matching on):

- The primary key.
- A non-key field or a subset of primary key fields (for example, two columns of a three-column primary key).

- A superset of the primary key (all key fields plus non-key fields).

In MODIFY, a MATCH on a partial key or on a non-key may retrieve more than one row. MATCH returns only the first row of this answer set; subsequent sections demonstrate how to use NEXT to retrieve the remaining rows.

In MAINTAIN, MATCH always matches on the full primary key and retrieves at most one row. To match on a partial key or non-key in MAINTAIN, you use the NEXT command without a prior MATCH. The MAINTAIN implementation of the NEXT command fetches the entire answer set returned by the RDBMS directly into a stack. It also includes three optional phrases:

- The FOR phrase determines how many rows to retrieve.
- The WHERE phrase defines retrieval criteria.
- The INTO phrase names a stack to receive the returned rows.

See the *FOCUS for IBM Mainframe MAINTAIN User's Manual* for complete syntax.

The Role of the Primary Key

In a table, the primary key is the column or combination of columns whose values uniquely identify a row in a table. Such columns may not contain null data.

The Master and Access Files for a table identify its primary key. Chapter 4, *Describing Tables to FOCUS*, explains how to describe primary key columns.

You can modify tables without primary keys as well as those with primary keys. Tables that lack primary keys have the attribute KEYS=0 in their Access Files. *Modifying Tables Without Primary Keys* on page 12-21 explains how to maintain tables without primary keys.

In DB2 and SQL/DS, you can implement RDBMS referential integrity by defining primary and foreign keys in SQL CREATE TABLE statements (see *Referential Integrity* on page 12-21). Defining the primary key to the RDBMS is optional. Most tables have primary keys (and unique indexes created to support them) whether or not the CREATE TABLE statement explicitly identifies them. In this chapter, the term **primary key** or **key** refers to those columns that compose each row's unique identifier.

Index Considerations

Indexes enhance the performance of data maintenance routines, especially indexes created on the table's primary key. Without an index, the RDBMS must read the entire table to locate particular rows; with an index, the RDBMS can access rows directly when given search values for the indexed columns. A table can have several associated indexes. Indexes created for performance reasons can be unique or non-unique. To use either RDBMS or FOCUS referential integrity, create indexes on foreign keys.

You can define a unique index on one or more columns in a table. When an index is unique, the concatenated values of the indexed columns in one row cannot be duplicated in any other row. A unique index is generally defined on a primary key. Once you create it, the RDBMS automatically prevents the insertion of duplicate index values. Any attempt to insert a duplicate row generates an error message.

An example of a unique index on a primary key is the employee ID (EMP_ID) in the sample EMPINFO table. Since no two employees can have the same employee number, the value in the EMP_ID column makes each row unique:

```
EMP_ID     LAST_NAME    FIRST_NAME
---------  -----------  ----------

111111111  SMITH        JON
123456789  JONES        ROBERT
222222222  GARFIELD     THOMAS
234567890  SMITH        PETER
```

You cannot add another row with EMP_ID 111111111 to this table.

Modifying Data

With the FOCUS MODIFY and MAINTAIN facilities, you can add new rows to a table, update column values for specific rows, or delete specific rows.

The Interface processes a MODIFY or MAINTAIN transaction with the following steps:

1. FOCUS reads the transaction for incoming data values.
2. The Interface generates the appropriate SQL SELECT statement.
3. The RDBMS either returns an answer set consisting of one or more rows that satisfy the SELECT request, or determines that the row does not exist.
4. After the RDBMS returns the answer set and/or return code, the Interface either
   - Performs the update operation (UPDATE or DELETE) on the returned answer set. With MODIFY, the Interface processes one row at a time; with MAINTAIN, it can either process one row at a time or a set of rows.
   - Creates the new row (INCLUDE). In MAINTAIN, it may create multiple rows.
5. The RDBMS changes the database appropriately.

In MODIFY, you must use the NEXT statement to process a multi-row answer set one row at a time. Each NEXT statement puts you physically at the next logical row in the answer set. In MAINTAIN, one NEXT command can process a multi-row answer set without a prior MATCH.

The MATCH Statement

In response to a MATCH statement, the Interface selects the first row in the table that meets the MATCH criteria.

The MATCH statement compares incoming data with one or more field values and then performs actions that depend on whether or not a row with matching field values exists in the table.

The syntax of the MATCH statement in MODIFY is

```
MATCH  field1 [ field2...fieldn ]
  ON MATCH action_1
  ON NOMATCH action_2
```

where:

fieldn
: Are fields representing columns. FOCUS compares incoming data values against existing column values. The fields can be any combination of key and/or non-key fields. Specify complete fieldnames; MATCH does not support truncated names.

action_1
: Is the operation to perform when a row’s values match the incoming data values.

action_2
: Is the operation to perform when a row’s existing values do not match the incoming data values.

The *FOCUS for IBM Mainframe User’s Manual* discusses these actions in detail.

MATCH processing for multi-table Master Files is the same as for a multi-segment FOCUS database.

See the *FOCUS for IBM Mainframe MAINTAIN User’s Manual* for MATCH syntax in MAINTAIN.

Acceptable actions for MATCH statements fall into eight groups. They are operations that:

- Include, change, or delete rows.
- Control MATCH processing, such as rejecting the current transaction.
- Read incoming data fields.
- Perform computations and validations, or type messages to the terminal.
- Control Case Logic.
- Control multiple-record processing.
- Activate and deactivate fields in MODIFY.
- Permanently store data in the RDBMS.

Interface MATCH Behavior

In MODIFY requests, there are two major differences in the way MATCH statements function for the Interface and for native FOCUS:

- With the Interface, you can change the value of a table's primary key (subject to RDBMS limitations) using the UPDATE statement. When modifying a FOCUS database, you cannot change key field values.
- You can MATCH on *any* field or combination of fields in the row. However, if the full primary key is not included in the MATCH criteria, the Interface may retrieve more than one row as a result of the MATCH.

  For example, if the primary key is EMP_ID and the incoming value for MATCH LAST_NAME is SMITH, the answer set contains all rows with last name SMITH.

Note: In MAINTAIN, MATCH functions identically for the Interface and for native FOCUS.

Example Using the MODIFY MATCH Statement

Consider a MODIFY request that maintains the EMPINFO table. It prompts for an employee ID and for a new salary; then it processes the incoming data. The annotated request contains the following MATCH statements:

```
    MODIFY FILE EMPINFO
    PROMPT EMP_ID CURRENT_SALARY
1.  MATCH EMP_ID
2.  ON MATCH UPDATE CURRENT_SALARY
3.  ON NOMATCH REJECT
    DATA
```

The incoming transaction contains the following values:

```
EMP_ID = 123456789
CURRENT_SALARY = 20000
```

The request processes as follows:

1. The MATCH statement compares the value of the incoming EMP_ID, 123456789, to the EMP_ID values in the rows of the EMPINFO table. Since EMP_ID is the primary key of this table, the RDBMS can return at most one row as a result of this MATCH.
2. If a row exists for EMP_ID 123456789, the MATCH statement updates the CURRENT_SALARY value of that row with the incoming value 20000.
3. If no row exists for EMP_ID 123456789, the MATCH statement rejects the transaction.

In MAINTAIN, you do not have to include an ON NOMATCH statement in order to reject a transaction; MAINTAIN automatically rejects a transaction that does not satisfy the MATCH criteria.

The NEXT Statement

In MODIFY, the NEXT statement provides a flexible means of processing multi-row answer sets by moving the current position in the answer set from one row to the next.

The syntax is

```
NEXT field
    ON NEXT action_1
    ON NONEXT action_2
```

where:

field
: Is any field in the table. It does not have to be a primary key.

action_1
: Is the operation to perform when there is a subsequent row in the answer set. May be any of the acceptable actions listed for MATCH in *The MATCH Statement* on page 12-5.

action_2
: Is the operation to perform when no more rows exist in the answer set.

The KEYORDER parameter in the Access File controls the sort order (by primary key) for NEXT. It determines whether to retrieve primary key values in low (ascending order) or high (descending order) sequence. *KEYORDER* in Chapter 4, *Describing Tables to FOCUS*, explains how to specify the KEYORDER parameter. The default is to sort by primary key in ascending order.

Your choice of MATCH and NEXT statement combinations determines the contents of the answer set. Subsequent sections explain these choices in more detail:

- NEXT statement without a MATCH statement. The Interface requests the retrieval of all rows in the table sorted by primary key.
- MATCH with the primary key or a superset of primary key columns. The MATCH returns the single row that is the starting point for any subsequent NEXT statements.
- MATCH on a non-key field or a subset of primary key columns. The RDBMS returns a multi-row answer set in which each row satisfies the MATCH criteria.

You can also use NEXT statements with multi-table structures (FOCUS views) to modify or display data in either Case Logic or non-Case Logic requests. If your MATCH or NEXT specifies a row from a parent table in a multi-table structure, that row becomes the current position in the parent table. A subsequent NEXT on a field in a descendant of that table retrieves the first descendant row in the related table. In MODIFY:

- Without Case Logic, you can retrieve all parent rows in the table and only the first descendant row of any specified related table.

- With Case Logic, you can retrieve all rows for each table defined in a multi-table Master File. To do so, first MATCH on the parent. Then, in another case, use NEXT to loop through the related tables (at the lowest level) until there are no more related instances. On NONEXT, return to the parent case for the next parent instance.

  You can trace Case Logic with the FOCUS TRACE facility. To invoke the TRACE facility for the Interface, include the TRACE command on a separate line after the MODIFY FILE statement. For complete information about the FOCUS TRACE facility, see the *FOCUS for IBM Mainframe User's Manual*. You can also use the Interface FSTRACE facilities described in Appendix D, *Tracing Interface Processing*.

The following sections illustrate different combinations of MATCH and NEXT statements with annotated examples. The MODIFY requests have been kept simple for purposes of illustration; you can create more sophisticated procedures. Assume KEYORDER=LOW for all of the examples.

Note: In MAINTAIN:

- The syntax of the NEXT command includes optional FOR and WHERE phrases that control the number of rows retrieved into a stack. As in MODIFY, the KEYORDER attribute in the Access File determines whether NEXT returns rows in ascending or descending order of the primary key.
- NEXT always starts its retrieval at the current database position; it will not retrieve a row it has already passed in its retrieval path unless you use the REPOSITION command to reset the current position.

  Also as in MODIFY, once you MATCH on a parent segment, a subsequent NEXT on a child segment retrieves descendant rows within the parent established by the MATCH. However, one NEXT statement can retrieve all such child instances, without Case Logic.
- The UPDATE, DELETE, and INCLUDE commands also incorporate the optional FOR phrase to process multiple rows from a stack. The system variables FOCERROR and FOCERRORROW inform you whether the entire set of rows processed successfully and, if not, which row caused the problem.

For complete details, see the *FOCUS for IBM Mainframe MAINTAIN User's Manual*.

NEXT Processing Without MATCH

If you use a NEXT statement without a previous MATCH statement in a MODIFY request, the RDBMS returns an answer set consisting of all rows in the table sorted by the primary key. Use the ON NEXT statement to view each row in ascending primary key order. In a MAINTAIN request, the FOR and WHERE phrases in the NEXT statement determine the number of rows retrieved, sorted by the primary key in KEYORDER sequence.

Example Using NEXT Without MATCH in MODIFY

In this MODIFY example, the NEXT command retrieves each row in ascending order of employee ID number (EMP_ID):

```
MODIFY FILE EMPINFO
NEXT EMP_ID
  ON NEXT TYPE "EMPLOYEE ID: <D.EMP_ID LAST NAME <D.LAST_NAME "
  ON NONEXT GOTO EXIT
DATA
END
```

The TYPE statements display the following on the screen:

```
EMPLOYEE ID: 071382660  LAST NAME STEVENS
   .
   .
   .
   .
EMPLOYEE ID: 222222222  LAST NAME GARFIELD
EMPLOYEE ID: 234567890  LAST NAME SMITH
EMPLOYEE ID: 326179357  LAST NAME BLACKWOOD
EMPLOYEE ID: 451123478  LAST NAME MCKNIGHT
EMPLOYEE ID: 543729165  LAST NAME GREENSPAN
EMPLOYEE ID: 818692173  LAST NAME CROSS
```

Example Using NEXT in MAINTAIN

The following MAINTAIN procedure retrieves the same answer set into a stack named INSTACK and displays the retrieved values on a Winform named WIN1 (consult the *FOCUS for IBM Mainframe MAINTAIN User's Manual* for instructions on creating Winforms):

```
MAINTAIN FILE EMPINFO
FOR ALL NEXT EMP_ID INTO INSTACK
WINFORM SHOW WIN1
END
```

NEXT Processing After MATCH on a Full Key or on a Superset

In MODIFY, NEXT processing is identical for either a MATCH on a full primary key or a MATCH on a superset (full key plus a non-key field).

When the initial MATCH is successful, the RDBMS retrieves one row. This establishes the logical "position" in the table. The subsequent NEXT statement causes the RDBMS to retrieve all rows following the matched row in key sequence.

Example Using NEXT After MATCH on a Full Primary Key in MODIFY

The following is an example of NEXT processing after a MATCH on a full primary key, the EMP_ID field:

```
    MODIFY FILE EMPINFO
    CRTFORM LINE 1
    " PLEASE ENTER VALID EMPLOYEE ID </1"
1.  " EMP: <EMP_ID  "
2.  MATCH EMP_ID
        ON NOMATCH REJECT
3.      ON MATCH GOTO GETREST
    CASE GETREST
4.  NEXT EMP_ID
        ON NEXT CRTFORM LINE 10
        " EMP_ID: <D.EMP_ID   LAST_NAME: <D.LAST_NAME  "
        ON NEXT GOTO GETREST
5.      ON NONEXT GOTO EXIT
    ENDCASE
    DATA
    END
```

The MODIFY procedure processes as follows:

1. The user enters the employee ID for the search, 222222222.
2. The MATCH statement causes the RDBMS to search the table for the entered value. If no such row exists, the transaction is rejected.
3. If the specified value matches a value in the EMP_ID column of the table, the procedure branches to the GETREST case; it contains the NEXT statement.
4. The NEXT statement retrieves the next logical row based on the sequence of EMP_ID. If such a row exists, the procedure displays the values of the EMP_ID and LAST_NAME fields. It continues to display each row in order of the key field, EMP_ID.
5. If there are no more rows, the procedure ends.

The output after executing this MODIFY procedure is:

```
PLEASE ENTER VALID EMPLOYEE ID                         (line 1)

EMP: 222222222                                         (line 3)

EMP_ID:  234567890   LAST_NAME:  SMITH                 (line 10)

EMP_ID:  326179357   LAST_NAME:  BLACKWOOD             (line 10)

EMP_ID:  451123478   LAST_NAME:  MCKNIGHT              (line 10)

EMP_ID:  543729165   LAST_NAME:  GREENSPAN             (line 10)

EMP_ID:  818692173   LAST_NAME:  CROSS                 (line 10)
```

Because of the NEXT statement, all employees after 222222222 display one at a time on the screen. Notice that the rows are retrieved in key sequence.

Example Using NEXT on a Full Primary Key in MAINTAIN

The following MAINTAIN procedure retrieves the same answer set into a stack named EMPSTACK. Assume that when MAINTAIN displays the Winform called WIN1, the user enters the transaction value, 222222222, into a stack named TRANS and presses a PF key to invoke the NEXTRECS case:

```
MAINTAIN FILE EMPINFO
WINFORM SHOW WIN1
CASE NEXTRECS
  FOR ALL NEXT EMP_ID INTO EMPSTACK WHERE EMP_ID GT TRANS.EMP_ID
ENDCASE
END
```

NEXT Processing After MATCH on a Non-Key Field or Partial Key

In a MODIFY request processed by the Interface, you do not have to MATCH on the full set of key fields. You can match on a non-key field or partial key. (MAINTAIN always matches on the full primary key, regardless of which fields you specify in the MATCH command.)

When you MATCH on a non-key column or subset of key columns, multiple rows may satisfy the MATCH condition. The MATCH operation retrieves the first row of the answer set, and the NEXT command makes the remaining rows in the answer set available to the program in primary key sequence.

Example Using MATCH on a Non-Key Field in MODIFY

This annotated procedure is the previous MODIFY procedure altered to MATCH on the non-key field LAST_NAME. The NEXT operation retrieves the subsequent rows from the answer set:

```
    MODIFY FILE EMPINFO
    CRTFORM LINE 1
    "  PLEASE ENTER A LAST NAME </1 "
1.  "  LAST NAME: <LAST_NAME </1"
2.  MATCH LAST_NAME
        ON NOMATCH REJECT
3.      ON MATCH CRTFORM LINE 5
        " EMP_ID: <D.EMP_ID   LAST_NAME: <D.LAST_NAME "
4.      ON MATCH GOTO GETSAME
    CASE GETSAME
5.  NEXT LAST_NAME
        ON NEXT CRTFORM LINE 10
        " EMP_ID: <D.EMP_ID   LAST_NAME: <D.LAST_NAME "
        ON NEXT GOTO GETSAME
6.      ON NONEXT GOTO EXIT
    ENDCASE
    DATA
    END
```

The MODIFY procedure processes as follows:

1. The user enters the last name (LAST_NAME) for the search, SMITH.
2. The MATCH statement causes the RDBMS to search the table for all rows with the value SMITH and return them in EMP_ID order. If the value SMITH does not exist, the transaction is rejected.
3. If the incoming value matches a value in the table, the procedure displays the employee ID and last name. (This is the first row of the answer set.)
4. After displaying the row, the procedure goes to the GETSAME case; it uses NEXT to loop through the remaining rows in the answer set.
5. Instead of retrieving the next logical row with a higher key value as in the previous example, the procedure retrieves the next row in the answer set (all rows in the answer set have the last name SMITH). If any exist, they display on the screen in order of the key, EMP_ID.
6. When no more rows exist with the value SMITH, the procedure ends.

The output from this MODIFY procedure follows:

```
PLEASE ENTER A LAST NAME

LAST_NAME  smith

EMP_ID:  111111111   LAST_NAME:  SMITH



EMP_ID:  112847612   LAST_NAME:  SMITH
```

A line displays on the screen for each employee with the last name SMITH. Employee ID 111111111 is the result of the MATCH operation; employee ID 112847612 is the result of the NEXT operation.

Example Using NEXT on a Non-Key Field in MAINTAIN

The following MAINTAIN procedure retrieves the entire answer set into a stack named EMPSTACK. Assume that when MAINTAIN displays the Winform named WINA, the user enters the transaction value (SMITH) into the first row of a stack named TRANS and presses a PF key to invoke the NEXTRECS case:

```
MAINTAIN FILE EMPINFO
WINFORM SHOW WINA
CASE NEXTRECS
  FOR ALL NEXT EMP_ID INTO EMPSTACK WHERE LAST_NAME EQ TRANS.LAST_NAME
ENDCASE
END
```

INCLUDE, UPDATE, and DELETE Processing

While MATCH and NEXT operations in MODIFY can operate on primary key or non-key columns and return single or multi-row answer sets, the MODIFY statements INCLUDE, UPDATE, and DELETE must always identify the target rows by their primary key. Therefore, in MODIFY, each update operation affects at most one row. (In MAINTAIN, the FOR phrase in the update statement determines the number of rows affected.)

Example Updating Rows With MODIFY

For example, suppose you want to display all the employees in a department and increase certain salaries:

```
    MODIFY FILE EMPINFO
    CRTFORM LINE 1
    " PLEASE ENTER A VALID DEPARTMENT </1"
1.  " DEPARTMENT: <DEPARTMENT "
2.  MATCH DEPARTMENT
       ON NOMATCH REJECT
       ON MATCH CRTFORM LINE 10
3.     "EMP_ID: <D.EMP_ID   SALARY: <T.CURRENT_SALARY> "
4.     ON MATCH UPDATE CURRENT_SALARY
       ON MATCH GOTO GETREST
    CASE GETREST
5.     NEXT EMP_ID
       ON NEXT CRTFORM LINE 10
       " EMP_ID: <D.EMP_ID   SALARY: <T.CURRENT_SALARY> "
       ON NEXT UPDATE CURRENT_SALARY
       ON NEXT GOTO GETREST
6.     ON NONEXT GOTO EXIT
    ENDCASE
    DATA
    END
```

The MODIFY procedure processes as follows:

1. The user enters the department (DEPARTMENT) for the search, PRODUCTION.
2. The MATCH statement causes the RDBMS to search the table for the first row with the value PRODUCTION and return them in key sequence (EMP_ID). If none exists, the transaction is rejected.
3. If the supplied value matches a database value, the procedure displays it.
4. The procedure updates the salary field for the first retrieved row using the turnaround value from the CRTFORM. EMP_ID establishes the target row for the update.
5. Each time it executes the NEXT, the procedure retrieves the next row with the same department, PRODUCTION. It displays each one in EMP_ID order. It updates the salary field for each retrieved row with the turnaround value.
6. When no more rows exist for department PRODUCTION, the procedure ends.

Example Updating Rows With MAINTAIN

In MAINTAIN, you can use stack columns as turnaround values to update a table. The following annotated MAINTAIN request updates the same rows as the preceding MODIFY request:

```
   MAINTAIN FILE EMPINFO
1. WINFORM SHOW WIN1
2. CASE MATCHREC
     FOR ALL NEXT EMP_ID INTO EMPSTACK WHERE DEPARTMENT EQ VALSTACK.DEPARTMENT
   ENDCASE
3. CASE UPDSAL
      FOR ALL UPDATE CURRENT_SALARY FROM EMPSTACK
   ENDCASE
   END
```

The MAINTAIN processes as follows:

1. A Winform named WIN1 appears. Assume that it displays an entry field labeled DEPARTMENT (whose source and destination stack is called VALSTACK) and a grid (scrollable table) with columns EMP_ID and CURRENT_SALARY. See the *FOCUS for IBM Mainframe MAINTAIN User's Manual* for instructions on creating Winforms.
2. The user enters a DEPARTMENT value for the search and presses a PF key to invoke case MATCHREC. Case MATCHREC retrieves the rows that satisfy the NEXT criteria and stores them in a stack named EMPSTACK. The Winform displays the retrieved rows on the grid.
3. The user edits all the necessary salaries directly on the Winform grid and then presses a PF key to invoke case UPDSAL which updates all salaries.

RDBMS Transaction Control Within MODIFY

The Interface supports the Logical Unit of Work (LUW) concept defined by the RDBMS. An LUW consists of one or more FOCUS maintenance actions (UPDATE, INCLUDE, or DELETE) that process as a single unit. The maintenance operations within the LUW can operate on the same or separate tables.

The RDBMS defines a transaction as all actions taken since the application first accessed the RDBMS, last issued a COMMIT WORK, or last issued a ROLLBACK WORK.

Within a Logical Unit of Work, the RDBMS either executes all statements completely, or else it executes none of them. If the RDBMS detects no errors in any of the statements within the LUW:

- FOCUS issues a COMMIT WORK statement. The changes indicated by the updates within the transaction are recorded in the table.
- The RDBMS releases locks on the target data. (See *Isolation Levels* on page 12-33.)

- Database changes become available for other tasks.

In response to unsuccessful execution of any statement in the transaction, the Interface:

- Issues a ROLLBACK WORK command. Target data returns to its state prior to the unsuccessful transaction. All changes attempted by the statements in the transaction are backed out.
- Does not execute the remaining statements in the transaction.
- Releases locks on the target data.
- Discards partially accumulated results.

The RDBMS and the Interface provide a level of automatic transaction management but, in many cases, this level of management alone is not sufficient. FOCUS supports explicit control of RDBMS transactions with the commands SQL COMMIT WORK and SQL ROLLBACK WORK in MODIFY, and with the commands COMMIT and ROLLBACK in MAINTAIN.

Note: SQL COMMIT WORK and SQL ROLLBACK WORK are native SQL commands—commands that the Interface passes directly to the RDBMS for immediate execution. (You can issue SQL commands in MODIFY, but not in MAINTAIN.)—Do not confuse these commands with the FOCUS COMMIT WORK and ROLLBACK WORK commands that apply to FOCUS databases only. The Interface ignores COMMIT WORK and ROLLBACK WORK without the SQL qualifier.

With the default AUTOCOMMIT setting (see Chapter 10, *Thread Control Commands*), unless you specify SQL COMMIT WORK and/or SQL ROLLBACK WORK in your MODIFY procedure (or COMMIT and/or ROLLBACK in your MAINTAIN procedure), all FOCUS maintenance actions until the END statement constitute a single LUW. If the procedure completes successfully, the Interface automatically transmits a COMMIT WORK command to the RDBMS, and the changes become permanent. If the procedure terminates abnormally, the Interface issues a ROLLBACK WORK to the RDBMS, and the database remains untouched. Since locks are not released until the end of the program, a long MODIFY or MAINTAIN procedure that relies on the default, end-of-program COMMIT WORK can interfere with concurrent access to data. In addition, you may lose all updates in the event of a system failure.

Note:

- MAINTAIN does not respect the SET AUTOCOMMIT ON FIN command that postpones automatic COMMIT processing until the end of the FOCUS session (see Chapter 10, *Thread Control Commands*). An automatic COMMIT is issued at the end of *every* procedure. All called procedures issue a COMMIT before returning to the calling procedure. The COMMIT issued by a called procedure commits *all* uncommitted changes, even those in the calling procedure.

  For more information, consult the *FOCUS for IBM Mainframe MAINTAIN User's Manual*.

- You cannot use the FOCUS CHECK facility when updating a table. The Interface ignores the CHECK statement. You must use the SQL COMMIT WORK statement in MODIFY or the COMMIT statement in MAINTAIN.

  You can COMMIT after each transaction, or you can use a counter within the procedure to COMMIT after a set number of transactions. This technique can reduce some of the overhead associated with frequent COMMIT processing.
- You can also issue the AUTOCOMMIT ON CRTFORM command in MODIFY CRTFORM procedures. (See *The Change Verify Protocol: AUTOCOMMIT ON CRTFORM* on page 12-37.)
- The actions described in this section do not apply to non-fatal transaction level RDBMS errors, such as an attempt to include a row that violates an RDBMS uniqueness constraint. In such cases, although the SQLCODE is negative, processing continues normally to the next transaction record. Check the FOCERROR variable (discussed in *Using the Return Code Variable: FOCERROR* on page 12-19) to determine whether or not to ROLLBACK within the MODIFY procedure.

RDBMS Transaction Termination (COMMIT WORK)

The native SQL COMMIT WORK statement signals the successful completion of a transaction at the request of the procedure. Execution of a COMMIT statement makes changes to the tables permanent. The syntax in a MODIFY request is:

```
SQL COMMIT WORK
```

You can issue a COMMIT WORK as an ON MATCH, ON NOMATCH, ON NEXT, or ON NONEXT condition, after an update operation (INCLUDE, UPDATE, DELETE), or within cases of a Case Logic request.

Note: In MAINTAIN, you must use the MAINTAIN facility’s COMMIT command to transmit an SQL COMMIT WORK to the RDBMS.

Example **Using COMMIT WORK in a MODIFY Procedure**

A COMMIT WORK example using Case Logic follows:

```
CASE PROCESS
   CRTFORM
   MATCH field1 ...
     ON MATCH insert, update, delete, ...
   GOTO EXACT
ENDCASE
CASE EXACT
   SQL COMMIT WORK
   GOTO TOP
ENDCASE
```

The PROCESS case handles the MATCH, ON MATCH, ON NOMATCH processing. Then it transfers to CASE EXACT which commits the data, instructing the RDBMS to write the entire Logical Unit of Work to the database.

RDBMS Transaction Termination (ROLLBACK WORK)

The native SQL ROLLBACK WORK statement signals the unsuccessful completion of a transaction at the request of the procedure. Execution of a ROLLBACK statement backs out all changes made to the tables since the last COMMIT statement. The syntax in a MODIFY request is:

```
SQL ROLLBACK WORK
```

You can design a MODIFY procedure to issue a ROLLBACK WORK statement if you detect an error. For example, if a FOCUS VALIDATE test finds an inaccurate input value, you may choose to exit the transaction, backing out all changes since the last COMMIT. You can issue ROLLBACK WORK as an ON MATCH, ON NOMATCH, ON NEXT, or ON NONEXT condition, or within cases of a Case Logic request.

Note: In MAINTAIN, you must use the MAINTAIN facility's ROLLBACK command to transmit an SQL ROLLBACK WORK to the RDBMS.

The Interface automatically executes an SQL ROLLBACK WORK statement when you exit from a transaction early. For example, if you exit a CRTFORM without specifying some action, the Interface automatically issues a ROLLBACK WORK statement on your behalf.

The RDBMS automatically issues a ROLLBACK WORK statement in case of system failure or when it detects a fatal data error, such as a reference to a column or table that does not exist.

Example Using ROLLBACK WORK in a MODIFY Procedure

The following is an example of the ROLLBACK WORK statement using Case Logic:

```
ON NOMATCH CRTFORM ...
ON NOMATCH VALIDATE ...
   ON INVALID GOTO ROLLCASE
       .
       .
       .
CASE ROLLCASE
   SQL ROLLBACK WORK
   GOTO TOP
ENDCASE
```

Code the ROLLBACK WORK statement before a REJECT statement. FOCUS ignores any action following the rejection of a transaction, except for GOTO or PERFORM.

For example:

```
ON MATCH SQL ROLLBACK WORK
ON MATCH REJECT
```

Example **RDBMS Transaction Control**

Each time an employee's salary changes, the following example updates the salary in the EMPINFO table and posts a historical pay record for the new salary in the related PAYINFO table. To ensure that both updates complete or neither one does, the MODIFY procedure places both actions prior to a COMMIT WORK statement. If the descendant table is not processed, ROLLBACK WORK discards the whole logical transaction.

```
MODIFY FILE EMPPAY
COMPUTE DAT_INC=&YMD;
CRTFORM LINE 1
"</2 <25 MODIFY FOR SALARY CHANGE </2 "
"<20 ENTER THE EMPLOYEE ID <EMP_ID "
MATCH EMP_ID
  ON MATCH CRTFORM LINE 7
  "<D.FIRST_NAME <D.LAST_NAME CURRENT SALARY <T.CURRENT_SALARY> "
  "PLEASE CHANGE THE SALARY "
  ON MATCH UPDATE CURRENT_SALARY
  ON NOMATCH REJECT
MATCH DAT_INC
  ON NOMATCH COMPUTE
    SALARY=CURRENT_SALARY;
  ON NOMATCH INCLUDE
  ON NOMATCH SQL COMMIT WORK
  ON MATCH SQL ROLLBACK WORK
  ON MATCH REJECT
DATA
END
```

Using the Return Code Variable: FOCERROR

The RDBMS produces a return code, the SQLCODE, that reflects the success or failure of SQL statements; FOCUS stores this return code in the variable FOCERROR. You can test FOCERROR in a MODIFY or MAINTAIN procedure and take the appropriate action if you encounter a non-fatal error. A return code of 0 indicates successful completion of the last SQL command issued (either a "native" SQL command, such as SQL DELETE or SQL COMMIT WORK, or an SQL command generated by MODIFY or MAINTAIN).

Fatal error conditions, such as an inactive RDBMS machine, automatically terminate the FOCUS procedure. Non-fatal errors, such as an attempt to include a duplicate value for a unique index, allow the procedure to continue.

You can test the FOCERROR variable in FOCUS VALIDATE or IF statements to determine whether to continue or terminate processing. For example, if FOCERROR is -803, the INCLUDE or UPDATE operation failed in an attempt to include a value for a unique index. This condition might indicate the need to ROLLBACK the transaction or re-prompt the user for new input values.

For a list of common SQL return codes, see Appendix E, *SQL Codes and Interface Messages*. For a complete list, see the DB2 or SQL/DS Messages and Codes manual.

Using the Interface SET ERRORRUN Command

With SET ERRORRUN ON, MODIFY processing continues even when a serious error occurs, allowing applications to handle their own errors in the event that an RDBMS error is part of the normal application flow. Code this command explicitly within the MODIFY program, preferably in CASE AT START where it executes once.

Note: MAINTAIN does not support the SET ERRORRUN command.

The syntax is

```
CASE AT START
   SQL SET ERRORRUN {OFF|ON}
ENDCASE
```

where:

`OFF`

Stops MODIFY processing when the RDBMS detects a fatal error (for example, when it cannot find the table name). OFF is the default.

`ON`

Enables MODIFY processing to continue despite fatal errors. Test the value of FOCERROR to determine the desired action after an RDBMS call.

When SET ERRORRUN is ON, the MODIFY procedure reports the error but continues execution. The MODIFY code can then test the value of FOCERROR to determine the cause of the error and take appropriate action. Be careful in evaluating the contents of FOCERROR, as failure to respond to a negative SQLCODE can cause unpredictable errors in subsequent RDBMS or MODIFY processing.

SET ERRORRUN returns to its default setting of OFF at the end of the MODIFY procedure.

The DB2 Resource Limit Facility

The DB2 Resource Limit Facility, also known as the Governor, sets a limit on the resources a dynamic SQL query may use. All SQL queries generated by the Interface are dynamic. (For a discussion of static SQL, see Chapter 13, *Static SQL*.) A DB2 database administrator can set limits on application plans, individual users, or both.

If an SQL request generated by a MODIFY or MAINTAIN procedure fails because a resource limit has been reached, the Interface posts the DB2 SQLCODE -905 to the FOCERROR variable.

Since SQLCODE -905 does not terminate a procedure, you need not SET ERRORRUN ON to continue MODIFY processing.

Modifying Tables Without Primary Keys

Both DB2 and SQL/DS permit tables with duplicate rows. Such tables cannot possibly have a primary key, since no combination of column values can make a given row unique.

The Interface provides a way of maintaining tables with duplicate rows. You describe these tables to FOCUS with the attribute KEYS = 0 in the Access File, and the Interface invokes special SQL syntax to maintain them. It adds the "WHERE CURRENT OF cursor_name" clause to the SQL UPDATE and DELETE statements it generates from MODIFY procedures. (Cursor_name is an Interface-generated value.) This syntax causes the RDBMS to keep track of the rows being updated.

Note:

- MAINTAIN does not support modifying unkeyed tables.
- When the Access File describes a table with a primary key to FOCUS, the NEXT statement retrieves rows in a known, logical sequence, by primary key in ascending or descending order.

  When KEYS = 0, FOCUS does not request the rows in a particular order. As a result, rows retrieved with the NEXT command arrive unordered. Moreover, the RDBMS determines the retrieval sequence, and it may vary each time the MODIFY procedure executes.
- The FOCURRENT variable is not supported (see *The FOCURRENT Variable* on page 12-39); its value is always 0 for tables without primary keys. The AUTOCOMMIT ON CRTFORM option (discussed in *The Change Verify Protocol: AUTOCOMMIT ON CRTFORM* on page 12-37.) is not available for unkeyed tables.
- To guarantee the integrity of the row on which the select cursor is positioned, the Interface adds a "FOR UPDATE OF" clause to the SELECT statement. This places an update lock on the selected rows.

Referential Integrity

Since primary and foreign key values establish relationships between separate tables, it is important to maintain these values in a consistent manner throughout the database. The term *referential integrity* defines the type of consistency that should exist between foreign keys and primary keys.

Performance considerations usually make it preferable to have RDBMS indexes on both primary and foreign keys.

The following definitions help explain referential integrity:

- **Primary key** is the column or combination of columns whose value uniquely identifies a row within the table. None of the key columns can contain null values. A table can only have one designated primary key.

  The employee ID (EMP_ID) is the primary key in the sample EMPINFO table. The values for the EMP_ID field make each row unique, since no two employees can have the same identification number.

- **Foreign key** is a column or combination of columns in one table whose values are the same as the primary key of another table. The foreign key may be unique or non-unique, but its value must match a primary key value in the other table or be null.

  The employee ID (or WHO field) in the sample COURSE table is a foreign key. This field is similar to the primary key in the EMPINFO table in that it contains the employee ID of every employee who has taken a course. It contains multiple rows for those employees who have taken more than one course.

- **Referential integrity** describes the synchronization of these key field values:

  - A value must exist as a primary key before it can be entered as a foreign key. For example, a specific employee ID must exist in the EMPINFO table before a course can be added for that employee in the COURSE table (INCLUDE referential integrity).

  - If a primary key is deleted, all references to its value as a foreign key must be deleted, set to null, or changed to reflect an existing primary key value (DELETE referential integrity).

The RDBMS can define and enforce referential integrity rules (constraints).

FOCUS can also provide referential integrity for those tables described in a multi-table Master File. The following sections discuss both types of referential integrity constraints.

RDBMS Referential Integrity

The RDBMS provides the ability to define relationships between tables by embedding referential integrity constraints in the table definitions. The RDBMS prohibits data changes that violate the rules, and applications using the Interface respect these defined constraints.

Note: Tables you create with the FOCUS CREATE FILE command do not contain primary or foreign key definitions and, therefore, do not participate in RDBMS referential integrity. Of course, you can use the ALTER TABLE command to add primary and foreign key definitions to such tables.

Violation of RDBMS referential integrity rules results in a negative SQLCODE. The Interface posts this return code to the FOCERROR variable; your MODIFY or MAINTAIN procedure can test it.

Referential integrity violations do not terminate MODIFY or MAINTAIN procedures, so you need not use the FOCUS SET ERRORRUN ON command to continue MODIFY processing.

With RDBMS referential integrity in place, you do not need FOCUS referential integrity to invoke some level of automatic referential integrity support. You may wish to maintain your tables in separate Master Files and let the RDBMS take care of all referential integrity enforcement.

You may also choose to describe the tables as related (using multi-table Master and Access Files) and take advantage of FOCUS referential integrity.

If you use both FOCUS referential integrity and RDBMS referential integrity, the RDBMS referential integrity takes precedence in cases of conflict. Make sure you are familiar with the RDBMS referential integrity constraints on the tables involved as well as FOCUS referential integrity behavior. Check with your RDBMS database administrator for specific referential integrity constraints.

FOCUS Referential Integrity

The Interface provides some level of automatic referential integrity for tables described in a multi-table Master File.

The following sections describe the rules and techniques for ensuring or inhibiting FOCUS INCLUDE and DELETE referential integrity. The examples use the ECOURSE Master File, a multi-table description that relates the EMPINFO and COURSE tables. (See Appendix C, *File Descriptions and Tables*.)

FOCUS INCLUDE Referential Integrity

FOCUS MODIFY facility syntax provides automatic referential integrity for inserting new rows in a related set of tables. The following rules apply:

- You must describe the related set of tables in one multi-table Master File. The multi-table description establishes the relationship (an embedded join) based on the primary and foreign keys in the tables.
- The primary key rows belong to the parent table in the description. The foreign key rows belong to the related table in the description.

  With a multi-table Master File, you cannot add a related row (foreign key) using the FOCUS MODIFY facility unless the primary key value already exists. Therefore, a MODIFY procedure that inserts rows must MATCH on the parent table before adding a row in a related table.

Example Using FOCUS INCLUDE Referential Integrity

The following examples demonstrate referential integrity when adding new rows. The scenarios are:

1. Add a course for an employee *only* if data for the employee ID already exists.
2. The employee ID does not exist. Add both a new employee ID and a course.

A simple, annotated FOCUS MODIFY procedure for each scenario follows.

The first example adds course information only if a row already exists for the employee:

```
    MODIFY FILE ECOURSE
    CRTFORM LINE 2
    "ADD COURSE INFORMATION FOR EMPLOYEE  </1"
1.  "EMPLOYEE ID: <EMP_ID  </1 "
    "COURSE NAME: <CNAME          GRADE: <GRADE "
    " YEAR TAKEN: <YR_TAKEN     QUARTER: <QTR   "
2.  MATCH EMP_ID
3.      ON MATCH CONTINUE
4.      ON NOMATCH REJECT
5.  MATCH CNAME
        ON NOMATCH INCLUDE
        ON MATCH REJECT
    DATA
    END
```

The MODIFY procedure processes as follows:

1. The user enters the employee ID and information about the course taken. This constitutes the incoming transaction record.
2. The MATCH statement causes the RDBMS to search the table for an existing row with the specified employee ID.
3. If the employee row exists, the MODIFY continues to the next MATCH statement.
4. If no row in the EMPINFO table exists with the specified employee ID, MODIFY rejects this transaction and routes control to the top of the FOCEXEC.
5. MATCH CNAME causes the RDBMS to search the COURSE table for an existing row with the specified course for the employee ID located in Step 2. If no such row exists, the MODIFY adds a row in the COURSE table. If the course row already exists, the MODIFY rejects the transaction as a duplicate.

The second example adds a row to the EMPINFO table for the new employee and adds a course for that employee to the COURSE table. If the employee ID already exists, the procedure adds only the course information to the COURSE table:

```
    MODIFY FILE ECOURSE
    CRTFORM LINE 1
1.      "ID: <EMP_ID "
2.  MATCH EMP_ID
3.      ON NOMATCH CRTFORM LINE 2
        "       LAST: <LAST_NAME        FIRST: <FIRST_NAME </1 "
        "  HIRE DATE: <HIRE_DATE         DEPT: <DEPARTMENT "
        "        JOB: <CURR_JOBCODE    SALARY: <CURRENT_SALARY  </1"
        " BONUS PLAN: <BONUS_PLAN      ED HRS: <ED_HRS  </1"
        "COURSE NAME: <CNAME             YEAR: <YR_TAKEN  QTR: <QTR "
        "      GRADE: <GRADE "
        ON NOMATCH INCLUDE
4.      ON MATCH CRTFORM LINE 9
        "COURSE NAME: <CNAME         YEAR: <YR_TAKEN  QTR: <QTR "
        "      GRADE: <GRADE "
5.  MATCH CNAME
        ON NOMATCH INCLUDE
        ON MATCH REJECT
    DATA
    END
```

The MODIFY procedure processes as follows:

1. The user enters EMP_ID.
2. The MATCH statement causes the RDBMS to search the EMPINFO table for an existing row for the specified employee ID.
3. If the employee row does not exist, the user enters the data for both the employee and the specified course. The procedure adds a row to each table.
4. If the employee already exists, the user enters only the course data.
5. The MATCH CNAME statement causes the RDBMS to search the COURSE table for the specified course. If this course does not exist for this employee, the procedure adds it. If it does exist, the procedure rejects the transaction.

   Notice that the MATCH statement only identifies CNAME. FOCUS automatically equates the value of EMP_ID with WHO, part of the key to COURSE.

FOCUS DELETE Referential Integrity

FOCUS provides automatic referential integrity for deleting rows in a related set of tables. Just as with INCLUDE referential integrity, only tables described in a multi-table Master File invoke FOCUS DELETE referential integrity.

Note: An attempt to use FOCUS DELETE referential integrity in conjunction with tables that have an RDBMS ON DELETE RESTRICT constraint on the child segments produces an error condition; when FOCUS attempts to delete a parent segment, the RDBMS restriction takes precedence and prevents the deletion.

When you delete a parent row (primary key) in a MODIFY or MAINTAIN procedure, FOCUS automatically deletes all related rows (foreign keys) at the same time.

Example Using FOCUS DELETE Referential Integrity

For example, when you delete an employee from the EMPINFO table in the ECOURSE Master File, FOCUS also deletes all rows from the COURSE table that represent courses the employee has taken:

```
    MODIFY FILE ECOURSE
    CRTFORM LINE 2
    "DELETE EMPLOYEE AND ALL COURSES </1"
1.  "EMPLOYEE ID: <EMP_ID   "
2.  MATCH EMP_ID
        ON MATCH COMPUTE DOIT/A1 = 'N';
        ON MATCH CRTFORM LINE 6
3.      "EMPLOYEE TO BE DELETED: <D.EMP_ID </1"
        "         LAST NAME:   <D.LAST_NAME </1"
        "         FIRST NAME:  <D.FIRST_NAME </1"
        "         HIRE DATE:   <D.HIRE_DATE </1"
        "         DEPARTMENT:  <D.DEPARTMENT </1"
        "         JOB CODE:    <D.CURR_JOBCODE </2 "
        "IS THIS THE EMPLOYEE YOU WISH TO DELETE? (Y,N): <DOIT "
        ON MATCH IF DOIT EQ 'N' THEN GOTO TOP;
4.      ON MATCH DELETE
        ON NOMATCH REJECT
    DATA
    END
```

The MODIFY procedure processes as follows:

1. The user enters the employee ID.
2. The MATCH statement causes the RDBMS to search the EMPINFO table for an existing row with the specified employee ID.
3. If the row exists, the MODIFY displays information for verification purposes.
4. Once verified, FOCUS deletes the employee and all associated rows in both the EMPINFO and the COURSE tables. When FOCUS deletes a parent, it automatically deletes all associated related instances.

Inhibiting FOCUS Referential Integrity

You may not always want to enforce FOCUS referential integrity. Consider a relationship in which COURSE is the parent table that contains the primary key and EMPINFO is the related table that contains the foreign key. If you delete a course offering, you do not want to delete all employees who have taken the course.

To handle this problem, specify the parameter WRITE=NO in the Access File for the related (foreign key) table. This gives you the ability to modify the COURSE table without affecting the data in the EMPINFO table. You can still use the data in the EMPINFO table for browsing or lookup tasks. This technique bypasses FOCUS referential integrity.

Another technique is to COMBINE single tables rather than using a multi-table Master File. COMBINE of single tables does not invoke FOCUS referential integrity.

The MODIFY COMBINE Facility

Some applications require that you use a single input transaction to update several tables in the same MODIFY procedure. If the tables are not defined in the same Master File, you can use the COMBINE facility to modify them as if they are one.

Note: In MAINTAIN, you do not issue a COMBINE command to modify unrelated tables. Instead, you reference multiple tables in the MAINTAIN FILE statement. For example:

```
MAINTAIN FILES EMPINFO AND COURSE
```

You can maintain up to 63 tables in a single MODIFY procedure that operates on a COMBINE structure. The COMBINE limit is 16 Master Files; however, each Master File can describe more than one table, for a total of 64 per procedure, minus one for the artificial root segment created by the COMBINE command.

The COMBINE facility links multiple tables and assigns a new name to them so FOCUS can treat the tables as a single structure. Tables in a COMBINE structure can have different SUFFIX attributes, but you cannot combine a FOCUS database with anything except other FOCUS databases.

Note: In MAINTAIN, you can modify FOCUS databases and RDBMS tables in the same procedure.

When you issue a COMBINE command, the COMBINE structure remains in effect for the duration of the FOCUS session or until you enter another COMBINE command. Only one COMBINE structure can exist at a time, so each subsequent COMBINE command replaces the existing structure.

Do not confuse COMBINE with the dynamic JOIN command. You use JOIN to *report* from multiple tables that share at least one common field or for LOOKUP functions. With the COMBINE facility, you can *MODIFY* multiple tables. COMBINE is part of the MODIFY command; only the MODIFY and CHECK FILE commands process COMBINE structures. The FIND function also works in conjunction with COMBINE (see *The FIND Function* on page 12-32).

Note that COMBINE considers the component structures to be unrelated. Although RDBMS referential integrity is enforced, FOCUS referential integrity does not apply to a COMBINE of single-table Master Files. Your procedure should check for and enforce referential integrity, if necessary.

Syntax How to Use the COMBINE Command

The basic syntax for the COMBINE command is

```
COMBINE FILES  file1 [PREFIX pref1|TAG tag1] [AND]
   .
   .
   .
               filen [PREFIX prefn|TAG tagn] AS asname
```

where:

file1 - filen

Are the Master File names of the tables you want to modify. You can specify up to 16 Master Files.

pref1 - prefn

Are prefix strings for each file; up to four characters. They provide uniqueness for fieldnames. You cannot mix TAG and PREFIX in a COMBINE structure. Refer to the *FOCUS for IBM Mainframe User's Manual* for additional information.

tag1 - tagn

Are aliases for the table names; up to eight characters. FOCUS uses the tag name as the table name qualifier for fields that refer to that table in the combined structure. You cannot mix TAG and PREFIX in a COMBINE, and you can only use TAG if FIELDNAME is set to NEW or NOTRUNC.

AND

Is an optional word to enhance readability.

asname

Is the required name of the combined structure to use in MODIFY procedures and CHECK FILE commands.

Once you enter the COMBINE command, you can modify the combined structure.

How FOCUS Creates a COMBINE Structure

For example, the EMPINFO table contains employee number, last name, first name, hire date, department code, current jobcode, current salary, number of education hours, and bonus plan information. A second table, PAYINFO, is a historical record of the employee’s pay history. It contains the employee number, date of increase, percent of increase, new salary, and jobcode. (Appendix C, *File Descriptions and Tables*, provides the Master and Access Files for these two tables.)

Each time a salary changes, both the EMPINFO and PAYINFO tables must reflect the change. Since both tables need to share data entered for employee number, salary and jobcode, this application is appropriate for the COMBINE facility. You can update both tables at the same time without having to define multi-table Master and Access Files.

The following figures represent the tables as separate entities.

```
EMPINFO table          PAYINFO table

           EMPINFO                PAYINFO
01         S0          01         S0
**************         **************
*EMP_ID      **        *PAYEID      **
*LAST_NAME   **        *DAT_INC     **
*FIRST_NAME  **        *PCT_INC     **
*HIRE_DATE   **        *SALARY      **
*            **        *            **
***************        ***************
**************         **************
          EMPINFO                PAYINFO
```

To modify the tables simultaneously, issue the following sequence of commands at the FOCUS command level or in a FOCEXEC:

```
COMBINE FILES EMPINFO PAYINFO AS EMPSPAY
MODIFY FILE EMPSPAY
    .
    .
    .
```

In the following picture, generated by the CHECK FILE command, FOCUS defines a new segment, identified as SYSTEM99, to be the root segment of the combined structure. SYSTEM99 acts as the traffic controller for this structure; it is a virtual (artificial) segment. It counts as one segment towards the total of 64 segments allowed in the COMBINE structure.

```
check file empspay pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   3  ( REAL=    3  VIRTUAL=   0 )
  NUMBER OF FIELDS=    15  INDEXES=   0  FILES=     3
  TOTAL LENGTH OF ALL FIELDS=   95
 SECTION 01
              STRUCTURE OF SQLDS    FILE EMPINFO  ON 07/22/93 AT 09.54.27

          SYSTEM99
  01      S0
 **************
 *            **
 *            **
 *            **
 *            **
 *            **
 ***************
  **************
       I
       +-----------------+
       I                 I
       I EMPINFO         I PAYINFO
  02   I S0        03    I S0
 **************    **************
 *EMP_ID      **   *PAYEID      **
 *LAST_NAME   **   *DAT_INC     **
 *FIRST_NAME  **   *PCT_INC     **
 *HIRE_DATE   **   *SALARY      **
 *            **   *            **
 ***************   ***************
  **************    **************
          EMPINFO           PAYINFO
```

The COMBINE facility makes it easy to modify many files with the same transaction. For additional information regarding the COMBINE facility of MODIFY, refer to the *FOCUS for IBM Mainframe User's Manual.*

SET INCLUDE SUBTREE

Prior to FOCUS Version 7.0, the Interface used segment activation logic in MODIFY write operations that differed from that of standard FOCUS against FOCUS files.

In a multi-path file structure involving FOCUS files (including structures generated by the COMBINE command), INCLUDE or DELETE actions operate only on active segments in the subtree of the segment specified in the previous MATCH or NEXT command.

However, with external DBMS files, INCLUDE or DELETE actions would operate not only on active segments in the subtree of the segment specified in the MATCH or NEXT command, but also on any active segments to the right of that segment in the hierarchy (as displayed by the CHECK FILE PICTURE command).

In Version 7.0, the default Interface behavior is consistent with the standard FOCUS behavior.

In Version 6.8, the SQL SET INCLUDE SUBTREE subcommand institutes the standard FOCUS behavior for external DBMS files. The syntax is:

```
SQL SET INCLUDE SUBTREE
```

You must place the subcommand on the line immediately following the MODIFY command.

Note: In Version 7.0, if you need to invoke the old default behavior for upward compatibility reasons, use the following subcommand on the line immediately following the MODIFY command:

```
SQL SET INCLUDE LATERAL
```

The LOOKUP Function

The LOOKUP function, used in FOCUS MODIFY procedures, retrieves data values from cross-referenced tables joined dynamically with the JOIN command. The function is valid in both MODIFY COMPUTE and VALIDATE statements.

The syntax for the LOOKUP function is

```
rfield/I1 = LOOKUP(field);
```

where:

rfield
: Contains the return code (1 or 0) after the LOOKUP function executes.

field
: Is the name of any field in a cross-referenced table. After the LOOKUP, this fieldname contains the field’s value for you to use as needed.

To use this feature most efficiently with an RDBMS, specify a cross-referenced field for which an RDBMS index has been established.

Note:

- The LOOKUP function is not supported between RDBMS tables and FOCUS databases in either direction.
- The extended syntax of the LOOKUP function (parameters GE and LE) is not valid for RDBMS tables. LOOKUP can only retrieve values that match exactly. Refer to the *FOCUS for IBM Mainframe User’s Manual* for more information.

The FIND Function

The FIND function, used with COMBINE structures in FOCUS MODIFY procedures or with any file in a MAINTAIN procedure, verifies the existence of a value in another file structure. The FIND function sets a temporary field to 1 if the value exists in the other file and to 0 if it does not. FIND does not return any actual data values. (See Appendix F, *Additional DB2 Topics*, for information on read-only access to IMS data from DB2 MODIFY procedures.)

Use FIND only with a table referenced in a COMBINE statement or a MAINTAIN FILE statement. With COMBINE, if the FIND is for a field in a VSAM file, this field must be the index or alternate index field. For MAINTAIN, the field must be indexed only if it is in a FOCUS database.

The syntax for the FIND function is

```
rfield/I1 = FIND(fieldname AS dbfield IN file);
```

where:

rfield

Contains the return code (1 or 0) after the FIND function executes.

fieldname

Is the comparison field from one COMBINE table or one table referenced in a MAINTAIN FILE statement.

dbfield

For MODIFY, is the field in another COMBINE file structure or an indexed field in a VSAM file, to use for the value comparison. The AS dbfield clause is optional if rfield and dbfield have the same name.

For MAINTAIN, is a fieldname from one of the files listed in the MAINTAIN FILE statement, qualified with its filename.

To use this feature most efficiently with an RDBMS, specify a field for which an RDBMS index has been established.

file

In MODIFY, names the table or VSAM file in which dbfield resides. In MAINTAIN, is ignored.

The FIND function is only supported within MODIFY or MAINTAIN procedures. For more information, consult the *FOCUS for IBM Mainframe User's Manual* or the *FOCUS for IBM Mainframe MAINTAIN User's Manual.*

Isolation Levels

While you are changing data in a table, the table is in an unstable state. The changes may eventually become permanent, or they may be backed out. In order for reports to contain meaningful results, users should not see changes until they are permanent.

To protect data integrity, the RDBMS must guarantee to the person updating a table that no other user will change the selected values before the first user's update is submitted.

The RDBMS provides a locking system for concurrency management. With native SQL, you can specify the length of time to hold a lock and, in SQL/DS, that the size of the lock be row rather than page. This section discusses the duration of the lock, called the **Isolation Level**.

During installation, every DB2 application plan or SQL/DS application package is bound with a default isolation level. The installation procedures for the Interface specify that the default isolation level is Cursor Stability. (Static MODIFY procedures are an exception. See Chapter 13, *Static SQL*, for information about Isolation Levels in static procedures.)

With the Cursor Stability setting, shared (read) locks on a data row or page are released as your cursor "moves" off that location. For instance, if a report reads many data pages, the shared lock acquired on each data page is released as the shared lock on the next data page is acquired.

Use Cursor Stability for read-only applications such as TABLE requests or read-only MODIFY or MAINTAIN procedures that browse data without updating. You may not use Cursor Stability with a MODIFY or MAINTAIN procedure that changes values in the database; doing so would leave the data susceptible to change by other tasks in the interim between the initial selection (MATCH) and the update or inclusion.

The second isolation level setting, Repeatable Read, provides the highest level of protection. With Repeatable Read, *any* lock acquired is held until the transaction boundary (COMMIT WORK or ROLLBACK WORK). Therefore, any data the MODIFY or MAINTAIN procedure displays on the screen remains unchanged until the procedure submits the update and executes a COMMIT WORK statement.

This higher level of protection is provided at the expense of concurrency, since all locks, including shared locks, remain in effect until the end of a Logical Unit of Work. Therefore, you should commit transactions on a regular basis throughout the MODIFY or MAINTAIN procedure. Periodic transaction termination is especially important for NEXT processing. All rows retrieved by NEXT remain locked even if they are not all updated. These retrieved rows are not available for update by another user until your procedure releases them (via SQL COMMIT WORK or SQL ROLLBACK WORK in MODIFY, or via COMMIT or ROLLBACK in MAINTAIN).

Because of these concurrency considerations, all read/write MODIFY and MAINTAIN procedures require the Repeatable Read isolation level, including those MODIFY procedures that invoke the Change Verify Protocol described in *The Change Verify Protocol: AUTOCOMMIT ON CRTFORM* on page 12-37.

In DB2 Version 4, there is an additional isolation level called Uncommitted Read (UR). It provides read-only access to records even if they are locked; however, these records may not yet be committed to the database. In DB2 5.1 two additional isolation levels have been added: NC and RS. For more information, see the *DB2 Command and Utility Reference.*

Note: The Interface AUTOCOMMIT ON CRTFORM feature is designed to eliminate some concurrency problems. (See *The Change Verify Protocol: AUTOCOMMIT ON CRTFORM* on page 12-37 for more information.) However, it also requires an isolation level of Repeatable Read.

The following sections describe how to change the isolation setting.

Changing the Isolation Level

You can change the isolation level by issuing the SET ISOLATION command in a MODIFY procedure or at the FOCUS command level. (For MAINTAIN, you must issue the SET ISOLATION command at the FOCUS command level prior to invoking the MAINTAIN procedure.) The setting remains in effect for the FOCUS session or until you reset it.

From the FOCUS command level, issue

```
SQL [target_db] SET ISOLATION level
```

where:

target_db
: Indicates the target RDBMS. Valid values are DB2 or SQL/DS. Omit if you previously issued the SET SQLENGINE command.

level
: Indicates the isolation level. Valid values are:

<u>CS</u>	Is Cursor Stability, the default. Releases shared locks as the cursor moves on in the table. Use for read-only requests.
RR	Is Repeatable Read. Use for MODIFY and MAINTAIN read/write routines. Locks the retrieved data until it is released by an SQL COMMIT WORK or SQL ROLLBACK WORK.
UR	Is Uncommitted Read. Available in DB2 only, Version 4 and higher. It provides read-only access to records even if they are locked; however, these records may not yet be committed to the database.
RS	Is Read Stability. Available in DB2 only, Version 5.1 and higher. For more information, see the *DB2 Command and Utility Reference.*
NC	Available in DB2 only, Version 5.1 and higher. For more information, see the *DB2 Command and Utility Reference.*
blank	A blank value resets the level to the Interface default.

Note:

- Omit the target RDBMS qualifier to issue the command in MODIFY procedures or if you previously issued the SET SQLENGINE command.
- The Interface does not validate the isolation level values. If you issue the SET ISOLATION command with a level not supported for the version of the RDBMS you are using, the RDBMS will return an SQL code of -104, signifying an SQL syntax error.
- For SQL/DS, the isolation level you set applies to locks held on tables in public dbspaces. (SQL/DS always locks private dbspaces at the dbspace level.)
- For DB2:
  - The SET ISOLATION command is enabled for only SELECT requests created as a result of FOCUS TABLE requests.
  - This setting cannot override the required isolation level of RR for MODIFY and MAINTAIN requests.

For more information about Interface commands, consult Chapter 11, *Environmental Commands*.

Changing the DB2 Isolation Level by Switching to Another Plan

The Interface SET PLAN command allows you, within a FOCUS session, to switch to a plan bound with a different isolation level. The SET PLAN command is only available if the DB2 Interface was installed using the Call Attachment Facility (CAF). (See Chapter 11, *Environmental Commands*, for other CAF-dependent commands.)

The systems group responsible for Interface installation must generate two DB2 application plans for the Interface, one bound with the Cursor Stability isolation level and the other with Repeatable Read. You can then use the Interface SET PLAN command to switch between the plans for the desired isolation level. After you issue the SET PLAN command, all FOCUS TABLE, MAINTAIN, or MODIFY procedures take advantage of the new isolation level.

In DB2 Version 4, there is an additional isolation level called Uncommitted Read (UR). It provides read-only access to records even if they are locked; however, these records may not yet be committed to the database. In DB2 5.1 two additional isolation levels have been added: NC and RS. For more information, see the *DB2 Command and Utility Reference*.

Issue the SET PLAN command from the FOCUS command level

```
SQL [DB2] SET PLAN planname
```

where:

planname
Is the name of your application plan; the default is DSQL unless your site changed the default at installation time.

Note:

- Omit the DB2 qualifier if you previously issued the SET SQLENGINE command for DB2.
- In non-CAF versions of the DB2 Interface, you must use a separate JCL procedure or CLIST to access each plan.

Issuing SQL Commands in MODIFY

With the Interface, you can execute a wide range of native SQL commands from within FOCUS. You can issue them in a MODIFY procedure or at the FOCUS command level.

In fact, you can execute all SQL commands that return only SQL return codes—not data—from within MODIFY. Since SQL SELECT returns data (rows), you cannot execute it from a MODIFY FOCEXEC. (You can, however, execute SQL SELECT requests at the FOCUS command level using the Direct SQL Passthru facility discussed in Chapter 9, *Direct SQL Passthru*.)

Note: MAINTAIN does not support SQL commands.

To place native SQL commands in a FOCUS MODIFY procedure, prefix them with the environmental qualifier SQL. Do not include TSO, MVS, or CMS environmental qualifiers or a target RDBMS qualifier. If the command exceeds one line, end the first line with a hyphen and continue the command on the next line, prefixing this continued line with the SQL qualifier. Semicolons are optional.

For example, the following section of a MODIFY procedure contains the SQL DELETE, CREATE TABLE, COMMIT WORK, and DROP TABLE statements:

```
CASE AT START
  SQL DELETE FROM PERSONNEL.TEMP1 WHERE ACCT_ID > 1000;
  SQL CREATE TABLE PERSONNEL.TEMP2 -
  SQL      (ACCT_ID INTEGER, AMOUNT DECIMAL (13,2)) -
  SQL       IN PUBLIC.SPACE0;
ENDCASE
    .
    .
    .
NEXT keyfield
  ON NEXT UPDATE anyfield
  ON NEXT SQL COMMIT WORK;
  ON NONEXT SQL DROP TABLE PERSONNEL.TEMP2;
    .
    .
    .
```

The SQL reference manual for the applicable RDBMS contains a comprehensive list of SQL commands.

The Change Verify Protocol: AUTOCOMMIT ON CRTFORM

The Interface AUTOCOMMIT ON CRTFORM facility provides application developers with an automated Change Verify Protocol to use as an alternative to the standard RDBMS method of concurrency and integrity in MODIFY CRTFORM procedures.

MAINTAIN does not support AUTOCOMMIT ON CRTFORM.

Without Change Verify Protocol, the Interface relies solely on the RDBMS to manage the concurrent update and retrieval activities of its applications. Using the SQL COMMIT WORK statement, application developers can define transactions, or Logical Units of Work (LUWs), consisting of one or more database actions. Within the LUW, the RDBMS guarantees database integrity; database objects manipulated by the LUW will not change in an unpredictable manner before the termination of the LUW. Furthermore, if any failure occurs within the boundaries of the LUW, the RDBMS will undo, or ROLLBACK, any outstanding updates within the LUW.

The RDBMS uses a locking mechanism to prevent other concurrent transactions from interfering with the LUW. The locking mechanism allocates and isolates database resources for the LUW. However, this approach suffers from at least one basic drawback: terminal I/O locks data for an indeterminate amount of time. The lock remains allocated to the LUW until the user chooses to react to the screen display. Database transaction throughput, in many cases, becomes more a function of RDBMS lock management than of RDBMS transaction processing performance.

AUTOCOMMIT ON CRTFORM automatically invokes the Change Verify Protocol through the following series of steps:

1. Before displaying a CRTFORM, the Interface issues an SQL COMMIT WORK statement to release all locks held on the underlying table. The COMMIT releases all locks for the record displayed on the terminal.
2. If the application requests an update to the displayed record (UPDATE, DELETE, or INCLUDE), the Interface retrieves the row from the table again. The Interface compares the held and newly-retrieved images of the transaction record to determine whether a conflict exists with a transaction from another user. If no conflict exists, FOCUS processes the update as expected. If another user changed the record in the interim, FOCUS rejects the update. The application should test the value of the FOCURRENT variable to redirect the logic flow in the FOCEXEC (see *The FOCURRENT Variable* on page 12-39).
3. Many applications retrieve a record and perform VALIDATE tests on that record. If the record satisfies those tests, the MODIFY branches to a case that matches the record again and (potentially) updates it. In this situation, AUTOCOMMIT ON CRTFORM retrieves and compares the displayed and current versions of the record. The second MATCH subcommand and the update request each sponsor the Change Verify Protocol action. If no conflict exists, FOCUS submits the update as expected. **Note:** In this scenario, the MODIFY application itself must be coded to check FOCURRENT (see *The FOCURRENT Variable* on page 12-39); the second MATCH does not automatically perform the check. Any maintenance action issued subsequent to the second MATCH subcommand still performs an automatic check.

The Change Verify Protocol does not have the unit-of-work capabilities of the RDBMS protocol, but it never holds locks on CRTFORM-displayed records. By eliminating locks from an application’s run-time path, it can substantially increase rates of transaction throughput and decrease terminal response times for interactive applications.

You can only issue the AUTOCOMMIT ON CRTFORM command in a MODIFY CRTFORM procedure. Place it in CASE AT START, not in the TOP case. You cannot switch AUTOCOMMIT modes within the MODIFY procedure.

The AUTOCOMMIT facility only works on single-record transactions. (For example, MODIFY MATCH and NEXT commands retrieve single records.) It is not designed for set processing with FOCUS multiple-record operations (REPEAT and HOLD, for example). For multiple-record processing, the Change Verify Protocol applies only to the last record.

The syntax is

```
SQL SET AUTOCOMMIT {OFF|ON CRTFORM}
```

where:

`OFF`
: Is the default; it retains the native RDBMS locking protocol.

`ON CRTFORM`
: Invokes the Change Verify Protocol.

To ensure data integrity in conjunction with AUTOCOMMIT ON CRTFORM, you must use an RDBMS isolation level of Repeatable Read (RR). See *Isolation Levels* on page 12-33 for a discussion of isolation levels.

The FOCURRENT Variable

A MODIFY procedure that invokes the Change Verify Protocol can test the value of the FOCURRENT variable to determine whether there is a conflict with another transaction. FOCUS stores a zero in FOCURRENT if there is no conflict, and the transaction is accepted. A non-zero value indicates a conflict; the transaction is rejected, an error message is displayed, and you can redirect the MODIFY activity.

The possible values for FOCURRENT are:

0	Transaction accepted
1	Invalid, record input will create duplicate
2	Invalid, record has been deleted
3	Invalid, record has been changed

Your MODIFY application should test FOCURRENT and branch according to its value. For example, a typical procedure using AUTOCOMMIT ON CRTFORM submits a transaction, tests FOCURRENT, and, if FOCURRENT is non-zero, resubmits the transaction.

Note: The FOCURRENT variable is not supported for tables that do not have primary keys; its value is always 0. The AUTOCOMMIT ON CRTFORM option is not available for unkeyed tables.

Rejected Transactions and T. Fields

FOCUS treats transactions rejected because of a conflict as if they failed a VALIDATE test. Transactions that use CRTFORM turnaround fields (T. fields) may require special handling in this case. If, within the logic of your application, you wish to re-retrieve a VALIDATE-rejected record from the database to display its current image, you must issue the MODIFY command DEACTIVATE INVALID. If you do not, the turnaround fields display the rejected values you attempted to enter. Decisions based on these values may be logically incorrect.

Note: MAINTAIN does not support field activation or deactivation.

For example, the following MODIFY request updates the CURRENT_SALARY field and contains a FOCURRENT test:

```
MODIFY FILE EMPINFO
-*
CRTFORM LINE 1
" EMP_ID  <EMP_ID "
GOTO EMPLOYEE
-*
CASE EMPLOYEE
  MATCH EMP_ID
  ON NOMATCH REJECT
  ON MATCH CRTFORM LINE 3
    " EMP_ID   <D.EMP_ID  "
    " SALARY   <T.CURRENT_SALARY> "
  ON MATCH UPDATE CURRENT_SALARY
  ON MATCH IF FOCURRENT NE 0 THEN GOTO UNDO; <== Check FOCURRENT and direct
ENDCASE                                          process for appropriate
-*                                               action.
CASE UNDO
  SQL ROLLBACK WORK
  DEACTIVATE INVALID
  GOTO EMPLOYEE
ENDCASE
-*
CASE AT START
  SQL SET AUTOCOMMIT ON CRTFORM  <== Releases record after display
ENDCASE
-*
DATA
END
```

In the following example, the CHANGE case (or second MATCH subcommand) applies the Change Verify Protocol action; the example also contains two VALIDATE tests:

```
MODIFY FILE EMPINFO
CRTFORM LINE 1
  " EMP_ID <EMP_ID "
GOTO VALIDATE
-*
CASE VALIDATE
  MATCH EMP_ID
  ON NOMATCH REJECT
  ON MATCH CRTFORM LINE 3
  " EMP_ID  <D.EMP_ID  "
  " SALARY  <T.CURRENT_SALARY> "
  " BONUS   <T.BONUS_PLAN>  "
  ON MATCH VALIDATE
    SALTEST= (CURRENT_SALARY GE 0) AND (CURRENT_SALARY LE 100000);
    BONTEST = (BONUS_PLAN GE 0) AND (BONUS_PLAN LE 100);
    ON INVALID TYPE "VALUES OUT OF RANGE "
    ON INVALID GOTO UNDO
  ON MATCH GOTO CHANGE
ENDCASE
-*
CASE CHANGE
  MATCH EMP_ID
    ON NOMATCH REJECT
    ON MATCH IF FOCURRENT NE 0 THEN GOTO UNDO;  <==   Check FOCURRENT and
    ON MATCH UPDATE CURRENT_SALARY BONUS_PLAN         direct process for
    ON MATCH IF FOCURRENT NE 0 THEN GOTO UNDO;  <==   appropriate action.
    ON MATCH SQL COMMIT WORK
ENDCASE
-*
CASE UNDO
  SQL ROLLBACK WORK
  DEACTIVATE INVALID        <== FOCUS does not return to old
  COMPUTE EMP_ID =EMP_ID;       screen with invalid data,
  GOTO VALIDATE                 but verifies data, then shows
ENDCASE                         refreshed data on the screen.
-*
CASE AT START
  SQL SET AUTOCOMMIT ON CRTFORM  <== Releases record after display
ENDCASE
-*
DATA
END
```

Note: If the application (and not the Interface) sponsors the second MATCH, the MODIFY application itself must check FOCURRENT (see *The FOCURRENT Variable* on page 12-39); the second MATCH does not automatically perform the check. Any maintenance action issued subsequent to the second MATCH subcommand still initiates an automatic check.

Loading Tables Faster With MODIFY: The Fastload Facility

The Interface Fastload Facility increases the speed of loading data into tables with MODIFY. It is especially effective when using FIXFORM to load large volumes of data into a table.

Use the following command to invoke the Fastload option from a MODIFY procedure:

```
SQL SET LOADONLY
```

For example:

```
MODIFY FILE table
SQL SET LOADONLY
        .
        .
        .
MATCH key1
    ON NOMATCH INCLUDE
    ON MATCH REJECT
        .
        .
        .
END
```

A standard FOCUS MODIFY procedure uses a MATCH statement to test for the existence of a single row whose primary key value matches that supplied by the input transaction. When the underlying table is an RDBMS table, the Interface uses an SQL SELECT statement to perform this test. After examining the return code resulting from the SELECT, FOCUS determines whether or not the row exists and directs MODIFY processing to the appropriate MATCH logic.

Fastload eliminates this SELECT operation. Its sole responsibility is to load rows into the RDBMS. Fastload does so without first determining whether the row already exists within the table. The RDBMS ensures the uniqueness of stored rows via its unique index. However, FOCUS messages about the existence of duplicates display online or go into a LOG file if one exists.

Fastload can only insert (ON NOMATCH INCLUDE) rows into a table. Any attempt to use other MODIFY maintenance functions produces a FOC1404 error message, and the procedure terminates.

Note: In the SQL/DS Interface, LOADONLY employs the SQL/DS blocked-insert statement, PUT. A non-zero return code from a PUT operation (such as a unique index violation) may result in failure to input more than one row. SQL/DS Interface applications using LOADONLY should test FOCERROR after every INCLUDE. If a non-zero code is returned, in all cases, applications should ROLLBACK and terminate the MODIFY procedure.

13 Static SQL

Topics:

- Requirements
- Creating a Static Procedure for DB2
- Plan Management in DB2
- Creating a Static Procedure for SQL/DS
- Using the Interface SQL COMPILE and SQL RUN Facilities
- Resource Restrictions

In prior releases of the Interface, FOCUS access to the RDBMS was entirely through dynamic SQL requests. While dynamic SQL is ideal for applications like ad hoc reporting, in which you do not know in advance what SQL statements you will execute at run time, it is less desirable for applications that do not require such flexibility.

In contrast, static SQL is ideal when you know beforehand all the SQL statements a procedure may execute. You register the procedure itself, including the SQL statements, with the RDBMS through a process known as binding (for DB2) or preprocessing (for SQL/DS). The resulting database object, called a DB2 application plan or package or an SQL/DS application package, is stored in the database and retrieved whenever you run the procedure. *The Interface as an RDBMS Application* in Chapter 1, *Introduction*, introduces bind concepts.

An important aspect of the static SQL process is that the RDBMS optimizes each SQL statement in the program and chooses an access path for it. A data access path consists of low-level data access requests that the RDBMS formulates and stores in internal format. When you execute the procedure, the RDBMS retrieves these requests and executes them immediately.

The net effect is that SQL statements in the static request are not reinterpreted at run time. The access path for each statement is preselected and reused each time you run the procedure.

Starting with FOCUS Version 7.0, the Interface provides a Static SQL facility for TABLE requests. (Static SQL for MODIFY has been available since FOCUS Version 6.8.) With the Static SQL for TABLE facility, you can generate static SQL modules for most FOCUS TABLE or MATCH FILE requests in any FOCEXEC. The FOCEXEC can also contain other FOCUS commands, such as MODIFY requests, SET commands, or Dialogue Manager logic; however, the Static SQL for TABLE facility translates only the TABLE and MATCH FILE requests into static SQL. To create static MODIFY procedures, you must use the Static SQL for MODIFY facility. Subsequent sections explain each facility in detail.

With the Static SQL facility, you can quickly and easily create application plans or packages (in DB2) or packages (in SQL/DS) for FOCUS procedures. When you run these procedures, you take advantage of the following security and performance benefits of static SQL:

- Procedure-level security.

  Privileges (for example, SELECT, UPDATE, or DELETE) for operations on a table are available only to authorized users of the application plan (or package) associated with the static procedure. Users have no access to underlying tables outside of the procedure. Under dynamic SQL, privileges must be granted on the actual tables. Many sites do not allow dynamic access to tables because users can then access the same tables using other programs, such as QMF™. Static SQL specifically addresses this security concern.

- Performance improvements.

  - Static SQL offers potentially substantial performance improvements for MODIFY procedures. The PREPARE cursor operation that checks syntax and authorization and selects access paths is removed from the run-time environment.

    MODIFY procedures tend to have many more SQL statements than TABLE requests, and each statement executed must first be prepared. Furthermore, the access path is lost each time the procedure executes a COMMIT or ROLLBACK WORK statement, even if the SQL statement has already been prepared. In procedures that COMMIT after each transaction, the same SQL statements must be prepared prior to every transaction. Many interactive programs COMMIT after each transaction to release locks and minimize the impact of a system or program failure. These programs benefit most from static SQL.

    If you do not normally compile your MODIFY procedures, you will probably notice that it takes less time to initialize the static procedure. FOCUS has already interpreted the MODIFY statements, and the time required to initialize the procedure is a function of compiled MODIFY rather than of static SQL.

  - FOCUS TABLE and MATCH FILE requests do not usually exhibit substantial performance improvement when converted to static processing. However, the SQL syntax checking, object authorization verification, and query optimization done when the static SQL module is bound, as opposed to at run time for a dynamic SQL request, generally result in slight performance improvements and decreased contention for the system catalog tables needed by the RDBMS to perform these operations. In general, the more SQL statements a procedure executes, the greater the performance improvement.

  Static SQL does not increase the actual speed of data retrieval or update. In a static SQL procedure, the time required to retrieve and/or update a given set of rows is identical to that of the same procedure executing dynamically, minus the cost of access path selection and authorization checking for the SQL statements.

Note: If RDBMS table statistics change, or if table or index structures are altered after the static procedure is bound, the procedure should be rebound to ensure optimum performance. Refer to *Optionally BIND the Plan for the FOCEXEC* on page 13-10 for a discussion of BIND in DB2; see *Execute the GENUSQL EXEC* on page 13-23 for corresponding information in SQL/DS. (*The Interface as an RDBMS Application* in Chapter 1, *Introduction*, includes a discussion of basic bind concepts.)

- Ease of Use.

  The FOCUS implementation of static SQL offers you a far less complex and time-consuming development cycle than is normally the case with static SQL applications. No third-generation language or SQL skills are required. In fact, you can write a FOCUS application to use static SQL without coding a single SQL statement. Additionally, you can develop, test, and revise your FOCEXEC using dynamic SQL without going through the cumbersome static SQL preparation process every time you make a change. You do not have to create the application plan or package until the end of the development cycle.

Requirements

You can implement many existing dynamic applications as static procedures with little preparatory work. A FOCEXEC does not require special coding (for instance, embedded SQL) to use static SQL. All current MODIFY procedures that update DB2 or SQL/DS tables (and can currently be compiled), and most current FOCEXEC procedures that access DB2 or SQL/DS tables using the TABLE FILE or MATCH FILE commands, can be registered as static SQL procedures with no alterations.

A MODIFY procedure to be compiled with the Static SQL for MODIFY facility can also contain SQL commands (for instance, SQL COMMIT WORK, INSERT, or UPDATE). A FOCEXEC to be compiled with the Static SQL for TABLE facility can contain SQL commands, MODIFY procedures, Direct SQL Passthru commands, and Dialogue Manager commands, although the Static SQL for TABLE facility does not convert these commands to static SQL. The FOCEXEC can also invoke other FOCEXECs; however, to run these additional FOCEXECs statically, you must convert them separately to static SQL.

You must satisfy the following requirements to create a static SQL procedure:

- A MODIFY procedure can contain only one MODIFY request. It cannot contain any other FOCUS or operating system statements. If the FOCEXEC contains Dialogue Manager statements, FOCUS prompts you for the values of any variables at compilation time. You cannot substitute new values at run time.

  A TABLE request can contain Dialogue Manager variables, and you can substitute new values when you execute the static procedure (see *SQL COMPILE and SQL RUN Processing* on page 13-37).

- All tables used in a MODIFY procedure must have primary keys. That is, you may not include any table whose FOCUS Access File specifies KEYS=0.

- A MODIFY procedure cannot use the WITH-UNIQUES method for processing unique segments.
- For DB2:
  - TSO ATTACH is not supported.

    The DB2 Interface must have been installed to use the Call Attachment Facility (CAF). You can verify this by using the SQL DB2 ? command to display Interface settings (Call Attachment Facility should be ON). There is no similar requirement for SQL/DS.
  - You cannot compile a MODIFY FOCEXEC as a static procedure if it accesses a DB2 view created with a GROUP BY or HAVING clause.

    The DB2 environment prohibits the type of static SQL syntax the Interface uses against a DB2 view created with a GROUP BY or HAVING clause. If you issue a static COMPILE for a FOCEXEC that operates against such a view, an SQLCODE of -815 results either at BIND or RUN time, depending on your current level of DB2 maintenance.
- For SQL/DS:

  The SQL/DS product does not support double-byte character set (DBCS) datatypes in static SQL procedures of the type generated by the Interface.

Note:

- Certain resource restrictions exist for any procedure that uses static SQL; under some circumstances they may make it impossible to COMPILE a MODIFY procedure to use static SQL. *Resource Restrictions* on page 13-42 explains these restrictions and how to determine whether your procedure may be affected.
- If your table contains a TEXT (LONG VARCHAR) field, place the description of that field at the end of the segment declaration for that table in the FOCUS Master File. No change has to be made to the RDBMS table. If you have more than one TEXT field in a table, place one of them last; it does not matter where you place the other TEXT field.
- The FOCCOMP library member for a MODIFY procedure contains a record of the fact that the procedure uses static SQL. To have both static and dynamic versions of the compiled procedure, you need two FOCCOMP libraries, one compiled with STATIC ON or NOBIND, and the other compiled with STATIC OFF.

Creating a Static Procedure for DB2

Creating a static TABLE request requires the Interface SQL COMPILE facility; its sole function is to create static SQL code for TABLE requests. Creating a static MODIFY is an extension to the usual procedure for compiling a FOCEXEC with the FOCUS COMPILE facility. Invoke the static creation process after you complete the FOCEXEC or MODIFY procedure.

You must allocate some additional DDNAMEs, you may need to issue the Interface SET STATIC command, and you must execute either the FOCUS COMPILE command (for MODIFY) or the Interface SQL COMPILE command (for TABLE). In response, the Interface automatically creates an Assembler program with the embedded SQL required by the procedure. It then precompiles, assembles, link-edits, and, optionally, binds the program. It accomplishes all steps, including the automatic BIND, from within FOCUS.

You must give some thought to the issue of plan management discussed in *Plan Management in DB2* on page 13-18 as it will affect your choice of BIND option.

This section outlines the steps required to create a static procedure, and provides a description of FOCUS processing for each step. *DB2 Static TABLE Example* on page 13-12 and *DB2 Static MODIFY Example* on page 13-15 provide annotated examples.

Creating a static module for DB2 consists of the following steps:

1. Write the procedure.
2. Allocate the required DDNAMEs.
3. Optionally issue the static SET command.
4. Optionally issue the SET SSID command.
5. Compile the FOCEXEC.
6. Optionally BIND the plan for the FOCEXEC.
7. Authorize users to run the plan.

Write the FOCEXEC

Virtually any TABLE or MATCH FILE request is valid. A MODIFY procedure, however, must be compilable. Consult *Requirements* on page 13-3 and the *FOCUS for IBM Mainframe User’s Manual* for more information on compiled MODIFY requirements.

Allocate the Required DDNAMEs

The following chart lists the DDNAMEs you must allocate before compiling a FOCEXEC to use static SQL. You can add these allocations to your FOCUS CLIST, batch JCL, or PROFILE FOCEXEC. Note that these allocations are additions to the normal allocations for running FOCUS with DB2:

DDNAME	**DCB Parameters**	**Description**

`ASMSQL`	`DSORG(PO)`
	`RECFM(FB)`
	`LRECL(80)`
		Target data set for the assembler source code generated by the compilation. The size of each member varies depending on the size of the TABLE requests in the FOCEXEC or the size of the MODIFY procedure and the number of columns in each table the MODIFY procedure references.

`DBRMLIB`	`DSORG(PO)`
	`RECFM(FB)`
	`LRECL(80)`
		Target data set for the database request module generated by the compilation.

`DB2LOAD`	n/a	DB2 load library. The name for this library is site-specific, but usually follows the form ‘DSNxy0.SDSNLOAD’ (where ‘xy0’ is DB2 Version x Release y). The DB2 load library should be the same one used by the DB2 subsystem where the BIND will take place. **Note:** The STEPLIB allocation of the DB2 load library is still necessary.

`STUBLIB`	`DSORG(PO)`
	`RECFM(U)`
		Contains the load module created for the procedure. This data set is also required at run time.

`SQLERR1`	`DSORG(PS)`
	`RECFM(FB)`
	`LRECL(130)`
		Contains the output of the IBM precompiler. DCB parameters are set by the precompiler. To route output to the terminal, use DA(*).

`SQLERR2`	`DSORG(PS)`
	`RECFM(FM)`
	`LRECL(121)`
		Contains the output of the IBM assemble operation. DCB parameters are set by the assembler. To route output to the terminal, use DA(*).

`SQLERR3`	`DSORG(PS)`
	`RECFM(FA)`
	`LRECL(81)`
		Contains the output of the IBM linkage editor. DCB parameters are set by the linkage editor. To route output to the terminal, use DA(*).

DDNAME	DCB Parameters	Description

`FOCCOMP`	`DSORG(PO)`
	`RECFM(VB)`
	`LRECL(32756)`
	`BLKSIZE(32760)`
		For MODIFY only. Contains the compiled MODIFY procedure. This data set is also required at run time.
		Note: The FOCCOMP library member for a MODIFY procedure contains a record of the fact that the procedure uses static SQL. If you have both static and dynamic versions of the compiled procedure, you need two FOCCOMP libraries, one compiled with STATIC ON or NOBIND, and the other compiled with STATIC OFF.

Optionally Issue the SET STATIC Command

The syntax for the static SET command for DB2 is

```
SQL [DB2] SET STATIC process
```

where:

`process`

Indicates the command that invokes static processing. Valid values are as follows:

`OFF`	Is the default setting. For MODIFY, reverses ON or NOBIND settings; does not invoke static processing for any subsequent COMPILE commands issued. For information about the FOCUS COMPILE command, see the *FOCUS for IBM Mainframe User's Manual*. For TABLE, invokes static processing with automatic BIND, which does not support extended plan management (see *Plan Management in DB2* on page 13-18) or modification of any of the default BIND parameters. If you require more flexibility, use the NOBIND option.
`ON`	Invokes static processing with automatic BIND. This option does not support extended plan management (see *Plan Management in DB2* on page 13-18) or modification of any of the default BIND parameters. If you require more flexibility, use the NOBIND option.
`NOBIND`	Invokes static processing without BIND. Use this option if you need extended plan management, to change the default BIND parameters, or to BIND your programs at another time (for example, during an off peak period).

Note:

- For TABLE requests, you do not have to issue the SET STATIC command unless you require the NOBIND option.
- If FOCUS performs the BIND, the Isolation Level is always CS for TABLE requests and RR for MODIFY requests. To change the Isolation Level, you must issue the BIND outside of FOCUS, as described in *Optionally BIND the Plan for the FOCEXEC* on page 13-10.
- Before requesting an automatic BIND, make sure the DB2 subsystem is operational. In addition, the BIND operation itself requires DB2 privileges that you may not possess. Your DB2 database administrator can tell you if you are authorized to use BIND.
- Omit the DB2 qualifier if you previously issued the SET SQLENGINE command for DB2.
- Batch considerations:

  You can create or run static procedures in batch. The DSN command processor cannot be invoked in batch; therefore, you must COMPILE your programs using the NOBIND option and BIND the programs separately.
- FOCUS Multi-Session Option (MSO) Considerations:

  You can create or run static procedures in MSO. The DSN command processor cannot be invoked from MSO; therefore, you must COMPILE your programs using the NOBIND option and BIND the programs separately.

Optionally Issue the SET SSID Command

You must issue the SET SSID command (see *SSID* in Chapter 11, *Environmental Commands*) if you request the automatic BIND option and the DB2 subsystem in which you want the BIND to occur is different from your site’s installation default. You can obtain the current setting for the DB2 subsystem by issuing the SQL DB2 ? command.

Note: The DB2 subsystem referenced by the SET SSID command should use the DB2 load library allocated to DDNAME DB2LOAD.

Compile the FOCEXEC

To compile a TABLE request, use the Interface SQL COMPILE facility; you must then use the Interface SQL RUN facility to run it.

To compile a MODIFY procedure, use the FOCUS COMPILE facility.

The SQL COMPILE and SQL RUN Commands for TABLE

The Interface provides facilities for creating and running static TABLE procedures. See *Using the Interface SQL COMPILE and SQL RUN Facilities* on page 13-34 for a complete description of the SQL COMPILE command, the SQL RUN command, SQL COMPILE and SQL RUN processing, host variable length considerations, and host variable placement.

FOCEXECs compiled with the Static SQL facility do not support the use of the LOAD command; to execute them, you must use the SQL RUN command.

The FOCUS COMPILE Command for MODIFY

Before compiling the request, be sure to allocate all input and output files, such as transaction files or log files, that the MODIFY will use at run time. Any COMBINE structures must be in effect both before compiling the MODIFY and at run time.

Issue the COMPILE command

```
COMPILE focexec
```

where:

focexec
Is the name of the MODIFY procedure to compile.

Starting with FOCUS Version 6.8, you do not have to recompile or rebind FOCUS applications for different releases of a FOCUS Version. This saves time and effort.

For example, a FOCUS static MODIFY procedure compiled and bound in Version 7.0 Release 8, does not have to be recompiled or rebound for FOCUS Version 7.0 Release 9, or vice versa. However, the procedure must be recompiled and rebound for a different FOCUS Version (for example, FOCUS Version 6.8).

Note: The ‘AS module’ extension to the COMPILE command is not supported for static SQL procedures. The module name must be identical to the FOCEXEC name.

FOCEXECs compiled with the Static SQL facility do not support the use of the LOAD command; to execute them you must use the RUN command.

Optionally BIND the Plan for the FOCEXEC

This step is only necessary if you issued the SET STATIC command with the NOBIND option.

You may choose to BIND one application plan for each FOCEXEC (basic plan management) or to combine several FOCEXECs into one application plan (extended plan management). Also use this option to supply BIND parameters (such as Isolation Level) other than the default. *Plan Management in DB2* on page 13-18 discusses plan management, and *The Interface as an RDBMS Application* in Chapter 1, *Introduction*, introduces bind concepts.

Issue the BIND command outside the FOCUS environment, using one of the methods supplied by IBM. Your BIND may include one or more FOCEXEC DBRMs.

For example:

```
BIND PLAN (BIGPLAN) MEM(FEX1 FEX2 FEX3 FEX4) ACTION(ADD) ISOLATION(CS)
```

For a full explanation of BIND methods and parameters, consult the *IBM DB2 Command and Utility Reference*.

Authorize Users to Run the Plan

Issue the SQL GRANT EXECUTE command to authorize users to execute the application plan created with the previous steps. This example shows how to issue the command from within a FOCUS session:

```
SQL DB2 GRANT EXECUTE ON PLAN BIGPLAN TO USER1, USER2
```

For more information on GRANT, consult the *IBM DB2 SQL Reference*.

Run-time Requirements

In addition to the allocations required for FOCUS and the Interface outlined in Chapter 2, *Getting Started*, you must include allocations for the FOCCOMP (for MODIFY only) and STUBLIB libraries allocated to DDNAMEs FOCCOMP and STUBLIB.

If you use extended plan management (see *Extended Plan Management* on page 13-19), you must issue the Interface SET PLAN command before running any of the procedures included in the DB2 application plan (see Chapter 12, *Maintaining Tables With FOCUS*):

```
SQL DB2 SET PLAN BIGPLAN
```

This setting overrides the plan for the FOCUS dynamic Interface. *Extended Plan Management* on page 13-19 explains options for resetting the plan at the conclusion of your procedures.

You can use the SQL DB2 ? command to view all current Interface plan settings, for example:

```
SQL DB2 SET PLAN BIGPLAN

SQL DB2 ?
>
    .
    .
    .
(FOC1448) ACTIVE PLAN FOR CALL ATTACH IS      - :
(FOC1459) USER SET PLAN FOR CALL ATTACH IS    - : BIGPLAN
(FOC1460) INSTALLATION DEFAULT PLAN IS        - : P7009411
    .
    .
    .
```

The most recently issued request to DB2 sets the Active Plan. If the thread is closed or marked inactive, no value displays for Active Plan. Your SET PLAN command determines the "User Set Plan".

Note: To execute a static MODIFY procedure, you do not need the Access File library (allocated to DDNAME FOCSQL) at run time if you will not be accessing DB2 dynamically during the FOCUS session. The Access File library is always required when executing a static TABLE FOCEXEC. The Master File library (allocated to DDNAME MASTER) is required for all static and dynamic access.

Processing and Security Overview

When you run a compiled FOCEXEC procedure, the Interface searches the STUBLIB load library for a load module of the same name and loads it, if it exists.

When you run a compiled MODIFY procedure, FOCUS loads the FOCCOMP library member into virtual memory. If the flag setting in this member indicates that this is a static procedure, the Interface searches the STUBLIB load library for a load module of the same name. The Interface compares timestamps in the FOCCOMP and STUBLIB members for consistency.

Next, FOCUS connects to DB2 and opens a thread to the application plan of the same name as the static procedure (you can use the SET PLAN command to override this). DB2 compares the owner, program name, and timestamps in the STUBLIB member and the DB2 application plan for consistency.

This process verifies that the compiled MODIFY (FOCCOMP member), the program load module (STUBLIB member), and the DB2 application plan are valid and that no substitutions have been made.

DB2 Static TABLE Example

The following annotated example illustrates the creation and automatic bind of a static FOCEXEC procedure named FEX1. It uses the following multi-table Master and Access File, EMPLOYEE, that relates two tables, PAYROLL and COURSES.

EMPLOYEE Master File:

```
FILENAME=EMPLOYEE,    SUFFIX=SQLDS,$
SEGNAME= PAYROLL,     SEGTYPE=S0,$
 FIELD=  EMP_ID,      EMP_ID,     A5,    A5, MISSING=OFF,$
 FIELD=  SALARY,      SALARY,     P13.2, P6, MISSING=OFF,$
SEGNAME= COURSES,     SEGTYPE=S0,$
 FIELD=  EMP_NUM,     EMP_NUM,    A5,    A5, MISSING=OFF,$
 FIELD=  COURSE_NUM,  COURSE_NUM, A5,    A5, MISSING=OFF,$
 FIELD=  POINTS,      POINTS,     I3,    I2, MISSING=OFF,$
```

EMPLOYEE Access File:

```
SEGNAME=PAYROLL, TABLENAME="USER1"."PAYROLL",
KEYS=1, WRITE=YES, KEYORDER=LOW,$
SEGNAME=COURSES, TABLENAME="USER1"."COURSES",
KEYS=2, WRITE=YES, KEYORDER=LOW,
KEYFLD=EMP_ID, IXFLD=EMP_NUM,$
```

Notice that the FEX1 FOCEXEC explicitly sets OPTIMIZATION, even though it requires the default value, ON:

```
SQL DB2 SET OPTIMIZATION ON
TABLE FILE EMPLOYEE
PRINT EMP_ID SALARY POINTS
WHERE SALARY GE &SALARY
END
```

The numbers 1 through 6 refer to the explanatory notes that follow the example. Also included for your information are the options FOCUS uses for each step (for example, Assembler options). Please refer to the appropriate IBM manual for an explanation of these parameters and their possible settings.

```
    SQL DB2 SET SSID DB2P

1.  SQL DB2 COMPILE FEX1 SALARY=50000
    END
2.  (FOC1472) STATIC SQL PROGRAM CREATED SUCCESSFULLY
3.  (FOC1473) STATIC SQL PROGRAM PREPROCESSED. RETURN CODE IS: 4
4.  (FOC1474) STATIC SQL PROGRAM ASSEMBLED. RETURN CODE IS  : 0
5.  (FOC1476) STATIC SQL PROGRAM LINKED. RETURN CODE IS  : 0
    DSN
6.  BIND PLAN(FEX1) MEM(FEX1) ACTION(REPLACE) ISOLATION(CS)
    DSNT252I -  BIND OPTIONS FOR PLAN FEX1
               ACTION        ADD
               OWNER         USER1
               VALIDATE      RUN
               ISOLATION     CS
               ACQUIRE       USE
               RELEASE       COMMIT
               EXPLAIN       NO
    DSNT253I -  BIND OPTIONS FOR PLAN FEX1
               NODEFER       PREPARE
               CACHESIZE     1024
               QUALIFIER     USER1
               CURRENTSERVER
               CURRENTDATA   NO
               DEGREE        1
               SQLRULES      DB2
               DISCONNECT    EXPLICIT
    DSNT200I -  BIND FOR PLAN FEX1     SUCCESSFUL
    DSN
    END
```

The steps in the process are:

1. Issue the SQL COMPILE command.

The user invokes the Static SQL for TABLE facility. The FOCEXEC includes a variable (&SALARY) to the right of a comparison operator in a screening condition. As described in *Using the Interface SQL COMPILE and SQL RUN Facilities* on page 13-34, when the SQL COMPILE command supplies a literal for a variable value, FOCUS substitutes an SQL host variable in that location. The value &SALARY=50000 in the SQL COMPILE command functions as a placeholder for the duration of the SQL COMPILE. The user must supply the actual value for the variable at SQL RUN time.

2. Create the Assembler program.

   In this step, FOCUS reads the FOCEXEC procedure and creates an Assembler program with embedded SQL statements representing all SQL statements the FOCEXEC will execute. This program is the input file for the next step.

3. Precompile the Assembler program.

   FOCUS invokes the IBM precompiler for the Assembler program created in Step 2. The precompiler comments out the embedded SQL statements and replaces them with Assembler calls to DB2. It places the SQL statements in a Database Request Module (DBRM) created as a member in the data set allocated to DBRMLIB. This member is part of the input to the BIND process. *The Interface as an RDBMS Application* in Chapter 1, *Introduction*, introduces bind concepts. FOCUS places the modified source program into a temporary data set. It writes the IBM precompiler listing to the data set allocated to DDNAME SQLERR1 or to the terminal.

   This step uses the Assembler precompiler included in the DB2 load library allocated to DDNAME STEPLIB. The Interface SET SSID command has no influence on the precompiler used. If you have more than one DB2 subsystem, be sure that the DB2 load library used in the precompilation step is the same one used by the DB2 subsystem in which the program will run. The DB2 subsystem does not have to be active during this step.

   The precompiler options for this step are HOST(ASM), DATE(ISO).

   Note: The precompilation process generates the statement “WARNINGS HAVE BEEN SUPPRESSED DUE TO LACK OF TABLE DECLARATIONS” in the Assembler program. This message is normal and may be ignored. The precompilation should complete with a return code of four (4).

4. Assemble the program.

   FOCUS assembles the modified source program using IBM's Assembler H and places the output into a temporary data set. It writes the resulting listing to the data set allocated to DDNAME SQLERR2 or to the terminal.

   The assembler options are DECK, NOOBJECT, NOALIGN and TERM. This step should complete with a return code of zero (0).

5. Link-edit the program.

   FOCUS links the assembled source program using IBM's linkage editor, placing the run-time module into the data set allocated to DDNAME STUBLIB. It writes linkage editor output to the data set allocated to DDNAME SQLERR3 or to the terminal.

   The linkage editor options are TERM, RENT, AMODE(31), RMODE(ANY), and LIST. This step should complete with a return code of zero (0).

6. Optional automatic BIND.

Since the request does not include the SET STATIC NOBIND command, FOCUS automatically invokes the DB2 DSN command processor and submits the following bind request:

```
DSN SYSTEM(DB2P)
BIND PLAN (FEX1) MEM(FEX1) ACTION(REPLACE) ISOLATION(CS)
```

Note:

- The SSID of the DB2 subsystem in which this plan will be bound is DB2P. The target subsystem for the BIND was established by the SET SSID command.
- The plan name will be the same as that of the FOCEXEC (FEX1). The MEM parameter means that application plan FEX1 will include member FEX1 from the data set allocated to DDNAME DBRMLIB. Member FEX1 was created during the precompilation step. The plan owner is the DB2 authorization ID in effect at the time of the bind.

FOCEXECs compiled with the Static SQL for TABLE facility do not support the use of the LOAD command; to execute them you must use the SQL DB2 RUN command.

DB2 Static MODIFY Example

The following annotated example illustrates the creation and automatic bind of a static MODIFY procedure named MOD1. The numbers 1 through 7 refer to the explanatory notes that follow the example.

Also included for your information are the options FOCUS uses for each step (for example, Assembler options). Please refer to the appropriate IBM manual for an explanation of these parameters and their possible settings.

```
    SQL DB2 SET SSID DB2P

    SQL DB2 SET STATIC ON

1.  COMPILE MOD1

    EMPLOYEE    ON 02/27/91 at 14.26.57
2.  (FOC1472) STATIC SQL PROGRAM CREATED SUCCESSFULLY:
3.  (FOC1473) STATIC SQL PROGRAM PREPROCESSED. RETURN CODE IS: 4
4.  (FOC1474) STATIC SQL PROGRAM ASSEMBLED   . RETURN CODE IS: 0
5.  (FOC1476) STATIC SQL PROGRAM LINKED      . RETURN CODE IS: 0

    DSN SYSTEM(DB2P)
6.  BIND PLAN(MOD1) MEM(MOD1) ACTION(ADD) ISOLATION(RR)
    DSNT252I - BIND OPTIONS FOR PLAN MOD1
               ACTION    ADD
               OWNER     PMSEF
               VALIDATE  RUN
               ISOLATION RR
               ACQUIRE   USE
               RELEASE   COMMIT
               EXPLAIN   NO
    DSNT253I - BIND OPTIONS FOR PLAN MOD1
               NODEFER         PREPARE
    DSNT200I -  BIND FOR PLAN MOD1     SUCCESSFUL
    END

7.  COMPILED...
```

The steps in the process are:

1. Issue the COMPILE command.

The user invokes the FOCUS COMPILE facility for MODIFY procedures.

2. Create the Assembler program.

In this initial step, FOCUS reads the MODIFY procedure and creates an Assembler program with embedded SQL statements representing all SQL statements the MODIFY will execute. This program is the input file for the next step.

3. Precompile the Assembler program.

   FOCUS invokes the IBM precompiler for the Assembler program created in Step 2. The precompiler comments out the embedded SQL statements and replaces them with Assembler calls to DB2. It places the SQL statements in a Database Request Module (DBRM) created as a member in the data set allocated to DBRMLIB. This member is part of the input to the BIND process. *The Interface as an RDBMS Application* in Chapter 1, *Introduction*, introduces bind concepts. FOCUS places the modified source program into a temporary data set. It writes the IBM precompiler listing to the data set allocated to DDNAME SQLERR1 or to the terminal.

   This step uses the Assembler precompiler included in the DB2 load library allocated to DDNAME STEPLIB. The Interface SET SSID command has no influence on the precompiler used. If you have more than one DB2 subsystem, be sure that the DB2 load library used in the precompilation step is the same one used by the DB2 subsystem in which the program will run. The DB2 subsystem does not have to be active during this step.

   The precompiler options for this step are HOST(ASM), DATE(ISO).

   Note: The precompilation process generates the statement “WARNINGS HAVE BEEN SUPPRESSED DUE TO LACK OF TABLE DECLARATIONS”. This message is normal and may be ignored. The precompilation should complete with a return code of four (4).

4. Assemble the program.

   FOCUS assembles the modified source program using IBM’s Assembler H and places the output into a temporary data set. It writes the resulting listing to the data set allocated to DDNAME SQLERR2 or to the terminal.

   The assembler options are DECK, NOOBJECT, NOALIGN and TERM. This step should complete with a return code of zero (0).

5. Link-edit the program.

   FOCUS links the assembled source program using IBM’s linkage editor, placing the run-time module into the data set allocated to DDNAME STUBLIB. It writes the linkage editor output to the data set allocated to DDNAME SQLERR3 or to the terminal.

   The linkage editor options are TERM, RENT, AMODE(31), RMODE(ANY), and LIST. This step should complete with a return code of zero (0).

6. Optional automatic BIND.

   Since the SET STATIC ON command requests an automatic BIND, FOCUS invokes the DB2 DSN command processor and submits the following bind request:

   ```
   DSN SYSTEM(DB2P)
   BIND PLAN (MOD1) MEM(MOD1) ACTION(REPLACE) ISOLATION(RR)
   ```

Note:

- The SSID of the DB2 subsystem in which this plan will be bound is DB2P. The target subsystem for the BIND was established by the SET SSID command.
- The plan name will be the same as that of the FOCEXEC (MOD1). The MEM parameter means that application plan MOD1 will include member MOD1 from the data set allocated to DDNAME DBRMLIB. Member MOD1 was created during the precompilation step. The plan owner is the DB2 primary authorization ID in effect at the time of the bind.

7. Compile the FOCEXEC.

FOCUS compiles the FOCEXEC and creates member MOD1 in the data set allocated to DDNAME FOCCOMP. It sets a flag in the FOCCOMP member to indicate that this is a static procedure.

FOCEXECs compiled with the Static SQL facility do not support the use of the LOAD command; to execute them you must use the RUN command.

Plan Management in DB2

Using extended plan management, you can create application plans that contain multiple procedures. This section discusses both the basic and extended plan management options. *The Interface as an RDBMS Application* in Chapter 1, *Introduction*, introduces application plans, packages, and bind concepts.

Basic Plan Management

With basic plan management, you create a separate DB2 application plan for every FOCEXEC or MODIFY procedure. You do not have to issue the SET PLAN command, and the Interface automatically establishes and terminates the necessary DB2 threads for each procedure invoked.

The following examples assume that MOD1 and MOD2 are static MODIFY procedures, that FEX1 and FEX2 are static FOCEXEC procedures, and that a corresponding DB2 application plan exists for each of them:

Step	MODIFY	FOCEXEC

1.	`RUN MOD1`	`SQL DB2 RUN FEX1`
		`END`

2.	`RUN MOD2`	`SQL DB2 RUN FEX2`
		`END`

3.	`RUN MOD2`	`SQL DB2 RUN FEX3`
		`END`

4.	`EX RPT1`	`EX RPT1`

The Interface takes the following actions:

1. First, it closes the thread to a prior application plan, if one exists. It deallocates that plan with a Call Attachment Facility (CAF) CLOSE.

   Then it opens a thread to plan MOD1 or FEX1. If no prior connection existed, it first issues a CAF CONNECT. It establishes the thread to MOD1 or FEX1 with a CAF OPEN.

2. It opens a thread to plan MOD2 or FEX2. First it deallocates the thread to MOD1 or FEX1 with a CAF CLOSE, then issues a CAF OPEN to establish the thread to MOD2 or FEX2.

3. No action. The Interface recalls that it already has a thread to MOD2 or FEX2 and takes no action.

4. It opens a thread to the FOCUS dynamic plan. First it deallocates the thread to MOD2 or FEX2 with a CAF CLOSE. Then it issues a CAF OPEN to establish the thread to the installation default application plan for the dynamic SQL Interface.

Note: The settings for AUTOCLOSE and AUTODISCONNECT affect thread retention across invocations of the same procedure, as in items 2 and 3. Some settings sever a thread and/or connection to DB2, either at the conclusion of any procedure, or within the procedure itself. More information is available in Chapter 10, *Thread Control Commands*.

Extended Plan Management

With extended plan management, you can create a DB2 application plan that contains more than one procedure. You must issue the SET PLAN command to establish an umbrella plan that includes each FOCEXEC to be invoked.

The primary purpose of extended plan management is to limit the amount of overhead involved in plan switching; therefore, this method of binding is not consistent with the AUTOCLOSE or AUTODISCONNECT options, both of which terminate threads and/or connections to DB2.

The following example assumes that MOD1 and MOD2, or FEX1 and FEX2, have been bound into a plan called BIGPLAN:

Step	**MODIFY**	**FOCEXEC**
1.	`SQL DB2 SET PLAN BIGPLAN`	`SQL DB2 SET PLAN BIGPLAN`

2.	`RUN MOD1`	`SQL DB2 RUN FEX1`
		`END`

3.	`RUN MOD2`	`SQL DB2 RUN FEX2`
		`END`

4.	`RUN MOD2`	`SQL DB2 RUN FEX3`
		`END`

5.	`SQL DB2 SET PLAN`	`SQL DB2 SET PLAN`
6.	`EX RPT1`	`EX RPT1`

The Interface takes the following actions:

1. It sets the application plan. The SET PLAN command tells the Interface that all required SQL statements for subsequent FOCEXECs are contained in BIGPLAN.
2. It opens a thread to plan BIGPLAN. It establishes the thread to BIGPLAN with a CAF OPEN. If there is no prior connection to DB2, it executes a CONNECT before the OPEN.
3. No action. The Interface recalls that the user-set plan is BIGPLAN and that the thread has already been established.
4. No action. The Interface recalls that the user-set plan is BIGPLAN and that the thread has already been established.
5. It returns control to the dynamic Interface. Setting the plan to blank instructs the Interface to return control to the installation default application plan. If you wish to return control to a plan other than the default plan, specify that plan name.

   Another option would be to include the dynamic Interface in BIGPLAN by including the DBRM for RSQL as part of the member list when binding BIGPLAN. This procedure would eliminate the need to reset the plan for dynamic access to DB2. If using this option, skip steps 5 and 6.
6. It opens a thread to the FOCUS dynamic plan. It executes a CAF OPEN to establish the thread to the installation default application plan.

Creating a Static Procedure for SQL/DS

Creating a static TABLE request requires the Interface SQL COMPILE facility; its sole function is to create static SQL code for TABLE requests. Creating a static MODIFY is an extension to the usual FOCUS procedure for compiling a FOCEXEC. Invoke the static creation process after you complete the FOCEXEC or MODIFY procedure.

To begin the static TABLE creation process, execute the Interface SQL COMPILE command (described in *Using the Interface SQL COMPILE and SQL RUN Facilities* on page 13-34). You begin the static MODIFY creation process by issuing the Interface SET STATIC command and executing the FOCUS COMPILE command.

In response, the Interface automatically creates an Assembler program with the embedded SQL required by the procedure. Complete the process by exiting FOCUS and running an Information Builders-supplied EXEC that preprocesses, assembles, and links the program and creates a package in the SQL/DS database.

This section outlines the steps required to create a static procedure, and provides a description of FOCUS processing for each step. *SQL/DS Static TABLE Example* on page13-26 and *SQL/DS Static MODIFY Example* on page 13-30 provide annotated examples.

Creating a static procedure for SQL/DS consists of the following steps:

1. Write the procedure.

2. For MODIFY, issue the static SET command.

3. Compile the FOCEXEC.

4. Exit FOCUS.

5. Execute the GENUSQL EXEC.

6. Authorize users to run the FOCEXEC.

Note: Execute all steps in this procedure after logging on to a userid authorized to connect to SQL/DS and after meeting the run-time requirements documented in Chapter 2, *Getting Started*.

Write the FOCEXEC

You can create a static procedure from virtually any TABLE or MATCH FILE request. The default Isolation Level is CS and cannot be changed.

A MODIFY procedure must be compilable. Consult *Requirements* on page 13-3 and the *FOCUS for IBM Mainframe User's Manual* for more information on compiled MODIFY requirements. The default Isolation Level is Cursor Stability (CS). If the MODIFY procedure updates any tables, include the SQL SET ISOLATION RR command in the MODIFY itself, immediately following the MODIFY FILE statement.

Issue the SET STATIC Command for MODIFY Procedures

To create a static MODIFY procedure, you may need to issue the SET STATIC command. Omit this step to create a static TABLE procedure.

The syntax for the static SET command for SQL/DS is

```
SQL [SQLDS] SET STATIC {OFF|ON}
```

where:

`OFF`
: Reverses ON setting. OFF is the default setting.

`ON`
: Invokes static processing.

Omit the SQLDS target database qualifier if you previously issued the SET SQLENGINE command for SQL/DS.

Compile the FOCEXEC

To compile a TABLE request, use the Interface SQL COMPILE facility; you must then use the Interface SQL RUN facility to run it. To compile a MODIFY procedure, use the FOCUS COMPILE facility.

The SQL COMPILE and SQL RUN Commands for TABLE

The Interface provides facilities for creating and running static TABLE procedures. Refer to *Using the Interface SQL COMPILE and SQL RUN Facilities* on page 13-34 for a complete description of the SQL COMPILE command, the SQL RUN command, SQL COMPILE and SQL RUN processing, host variable length considerations, and host variable placement.

FOCEXECs compiled with the Static SQL facility do not support the use of the LOAD command; to execute them, you must use the SQL RUN command.

The FOCUS COMPILE Command for MODIFY

Before compiling the request, issue FILEDEF commands for all input and output files, such as transaction files or log files, that the MODIFY will use at run time. Any COMBINE structures must be in effect both before compiling the MODIFY and at run time.

Issue the COMPILE command

```
COMPILE focexec
```

where:

`focexec`
: Is the name of the MODIFY procedure to compile.

Starting with FOCUS Version 6.8, you do not have to recompile or rebind FOCUS applications for different releases of a FOCUS Version. This saves time and effort.

For example, a FOCUS static MODIFY procedure compiled and bound in Version 7.0 Release 8, does not have to be recompiled or rebound for FOCUS Version 7.0 Release 9, or vice versa. However, the procedure must be recompiled and rebound for a different FOCUS Version (for example, FOCUS Version 6.8).

FOCEXECs compiled with the Static SQL facility do not support the use of the LOAD command; to execute them you must use the RUN command.

Note: The ‘AS module’ extension to the COMPILE command is not supported for static SQL procedures. The module name must be identical to the FOCEXEC name.

Exit FOCUS

Issue the FIN command to exit from FOCUS before running the GENUSQL EXEC.

Execute the GENUSQL EXEC

The GENUSQL EXEC preprocesses, assembles, links, and creates an application package for your static procedure. GENUSQL requires input parameters for execution; they acquire default values within the EXEC. To change the defaults at run time, provide overriding values on the command line in the form PARM=value. Use blanks as delimiters between parameter values; do not insert spaces on either side of the assignment symbol (=)

```
EX GENUSQL program [(parm=value parm=value] [DBLIST (db1 db2...dbn)]
```

where:

program

Is the name of the program to be prepared. It will have the same name as the MODIFY or TABLE FOCEXEC.

parm

Is one of the following:

`SQLUID`	Is the SQL/DS userid to be the creator of the package. Your VM logon ID is the default.
`SQLPSWD`	Is the SQL/DS password associated with SQLUID. This is not the VM logon password. No password is required for the VM userid default.
`COLLID`	Is the collection-ID, used as the qualifier for the SQL/DS application package produced by the SQLPREP EXEC. If omitted from the parameter list, it defaults to the SQLUID (if specified), or to the current VM userid (if SQLUID is not specified in the parameter list).
`GRANT`	Automatically grants run privilege on the package to PUBLIC. Possible values are YES or NO (the default).
`EXPLAIN`	Causes the SQL/DS optimizer to update the user's EXPLAIN tables with access path information. Possible values are NO, the default, or YES, to update the EXPLAIN tables.
`INDVAR`	Removes null indicator variables in SQL predicates in the ASMSQL program used as input to GENUSQL. The existence of these variables may degrade performance in applications involving large SQL/DS tables. Possible values are YES, the default, or NO (to remove the variables). **Note:** If you specify NO for INDVAR, no column referenced by a KEYS parameter or in a KEYFLD/IXFLD pair can allow nulls. Violation of this condition may cause unpredictable results. In addition, a setting of NO for INDVAR affects the total number of host variables to be applied towards the SQL/DS limit of 512 (SQL/DS Version 3 Release 3 and below).

db1...dbn

Is a list of databases. You can specify multiple databases as targets for storage of the application package. The default is the database specified in the most recently issued SQLINIT command.

Note:

- The default Isolation Level is Cursor Stability (CS). To invoke the Repeatable Read Isolation Level (RR) for a MODIFY procedure, include the SQL SET ISOLATION RR command in the MODIFY itself, immediately following the MODIFY FILE statement. You cannot change the default Isolation Level for a TABLE request.
- SQL/DS must be running when you invoke GENUSQL.

Authorize Users to Run the FOCEXEC

If you ran GENUSQL with GRANT=NO (the default), issue the SQL GRANT RUN command to authorize users to execute the packages created with the previous steps. The following example shows how to issue the command from within a FOCUS session:

```
SQL SQLDS GRANT RUN ON PLAN FEX1 TO USER1, USER2
```

For more information on GRANT, consult the *IBM SQL/DS SQL Reference*.

Run-time Requirements

In addition to the usual run-time requirements for the FOCUS SQL/DS Interface outlined in Chapter 2, *Getting Started*, make sure that the FOCCOMP files for any compiled MODIFY procedures are on a minidisk that you accessed prior to invoking FOCUS.

The STUBLIB LOADLIB containing the run-time modules must reside on a minidisk that you accessed prior to invoking FOCUS. It is not necessary to GLOBAL the STUBLIB LOADLIB, as the FOCUS EXEC automatically does this. However, you must access the minidisk before invoking FOCUS.

Processing and Security Overview

When you run a compiled FOCEXEC procedure, FOCUS searches for a module of the same name in the STUBLIB LOADLIB file.

When you run a compiled MODIFY procedure, FOCUS searches for a FOCCOMP file on one of the minidisks you accessed prior to invoking FOCUS. If the flag in the FOCCOMP file indicates that it is a static procedure, FOCUS searches the STUBLIB LOADLIB for a module of the same name. The Interface compares consistency tokens in the FOCCOMP and STUBLIB files.

FOCUS then invokes the static FOCEXEC or MODIFY procedure, connects to SQL/DS, and loads the package (which always has the same name as the FOCEXEC). SQL/DS verifies that the STUBLIB load module for the program and the SQL/DS package have the same owner, program name, and consistency token.

This process verifies that the compiled MODIFY (FOCCOMP file), the program load module (STUBLIB LOADLIB member), and the SQL/DS application package are valid and that no substitutions have been made.

Note: There is no equivalent in SQL/DS for the plan management techniques available for DB2. There must be a separate package for every static procedure, and one for the dynamic SQL/DS Interface. SQL/DS “switches” packages automatically in response to program execution.

SQL/DS Static TABLE Example

The following annotated example illustrates the creation of a static FOCEXEC procedure named FEX1. This example uses a multi-table Master and Access File, EMPLOYEE, that relates two tables, PAYROLL and COURSES. *DB2 Static TABLE Example* on page 13-12 provides the Master and Access Files.

The FEX1 FOCEXEC follows:

```
SQL SQLDS SET OPTIMIZATION ON
TABLE FILE EMPLOYEE
PRINT EMP_ID SALARY POINTS
WHERE SALARY GE &SALARY
END
```

The numbers 1 through 9 refer to the explanatory notes that follow the example. Also included for your information are the options FOCUS uses for each step (for example, assembler options). Please refer to the appropriate IBM manual for an explanation of these options and their possible settings.

```
1.  SQL SQLDS COMPILE FEX1 SALARY=50000
    END

2.  (FOC1472) STATIC SQL PROGRAM CREATED SUCCESSFULLY
3.  FIN
    Ready;
```

```
4.  genusql fex1 (sqluid=user1 sqlpswd=myword grant=yes
    Following installation parameters assumed:
          Pgm     = FEX1
          Sqluid  = USER1
          Sqlpswd = MYWORD
          Grant   = YES
          Indvar  = YES
          Collid  =
5.  ARI0717I START SQLPREP EXEC: 11/07/94 15:22:25 EST
    ARI0320I THE DEFAULT DATABASE NAME IS SQLDBA.
    ARI0663I FILEDEFS IN EFFECT ARE:
    SYSIN    DISK     FEX1     ASMSQL   *
    SYSPRINT DISK     FEX1     LISTPREP A1
    SYSPUNCH DISK     FEX1     ASSEMBLE A1
    ARISQLLD DISK     ARISQLLD LOADLIB  G1
    ARI0713I PREPROCESSOR ARIPRPA CALLED WITH THE FOLLOWING PARAMETERS:
    ........PREP=FEX1,USER=USER1/********,ISOL(USER),BLK,NOPR,DATE(ISO)
    ARI0708I ALL SQLPREP EXEC PROCESSING COMPLETED SUCCESSFULLY.
    ARI0796I END SQLPREP EXEC: 11/07/94 15:22:28 EST
6.  Assembling of FEX1 completed successfully.
7.  ARI0717I START SQLDBSU EXEC: 11/07/94 16:21:00 EST
    ARI0659I LINE-EDIT SYMBOLS RESET:
             LINEND=# LINEDEL=OFF CHARDEL=OFF ESCAPE=OFF TABCHAR=OFF
    ARI0656I MESSAGE FILE (SYSPRINT): TERMINAL
    ARI0655I INPUT FILE (SYSIN): TERMINAL
    ARI0320I THE DEFAULT DATABASE NAME IS SQLDBA.
    ARI0663I FILEDEFS IN EFFECT ARE:
    ARISQLLD DISK     ARISQLLD LOADLIB  G1
    SYSIN    TERMINAL
    SYSPRINT TERMINAL
    ARI0801I DBS Utility started: 11/07/94 16:21:05.
             AUTOCOMMIT = OFF ERRORMODE = OFF
             ISOLATION LEVEL = REPEATABLE READ
    ARI0828I ...LINEWIDTH reset to 80.
    ARI0827I ...Begin command execution: ERRORMODE = CONTINUE
    ARI0870I Enter the command terminated by semicolon or enter
             EXIT to end.
    ------>
    ------> CONNECT "USER1   " IDENTIFIED BY ********;
    ARI8004I User USER1 connected to server SQLDBA.
    ARI0500I SQL processing was successful.
    ARI0505I SQLCODE = 0 SQLSTATE = 00000 ROWCOUNT = 0

    ARI0870I Enter the command terminated by semicolon or enter
             EXIT to end.
    ------> GRANT RUN ON  USER1.FEX1 TO PUBLIC;
    ARI0500I SQL processing was successful.
    ARI0505I SQLCODE = 0 SQLSTATE = 00000 ROWCOUNT = 0

    ARI0870I Enter the command terminated by semicolon or enter
             EXIT to end.
    ------> COMMIT WORK;
    ARI0500I SQL processing was successful.
    ARI0505I SQLCODE = 0 SQLSTATE = 00000 ROWCOUNT = 0

    ARI0870I Enter the command terminated by semicolon or enter
             EXIT to end.
    ------> EXIT
    ARI8997I ...Begin COMMIT processing.
    ARI0811I ...COMMIT of any database changes successful.
    ARI0809I ...No errors occurred during command processing.
    ARI0808I DBS processing completed: 11/07/94 16:23:39.
    ARI0660I LINE-EDIT SYMBOLS RESTORED:
             LINEND=# LINEDEL=¢ CHARDEL=@ ESCAPE=" TABCHAR=ON
    ARI0796I END SQLDBSU EXEC: 11/07/94 16:23:39 EST
8.  DMSCPY721I Copy ARIRVSTC TEXT G1 APPEND FEX1 TEXT A1 (old file)
    IEW0000     INCLUDE OBJ
    IEW0000     ENTRY ADDRESS
9.  IEW0000     NAME FEX1(R)

    Ready;
```

The steps in the process are:

1. Issue the SQL COMPILE command.

The user invokes the Static SQL for TABLE facility. The FOCEXEC includes a variable (&SALARY) to the right of a comparison operator in a screening condition. As described in *Using the Interface SQL COMPILE and SQL RUN Facilities* on page 13-34, when the SQL COMPILE command supplies a literal for the variable value, FOCUS substitutes an SQL host variable in that location. The value &SALARY=50000 in the SQL COMPILE command functions as a placeholder for the duration of the SQL COMPILE. The user must supply the actual value for the variable at SQL RUN time.

2. Create the Assembler program.

In this step, FOCUS reads the FOCEXEC program and creates an Assembler program with all the embedded SQL statements required by the FOCEXEC.

The Assembler program has the same filename as the FOCEXEC and a filetype of ASMSQL. It is placed on the minidisk accessed as filemode A. In this example, the Assembler program is called FEX1 ASMSQL A.

3. Exit FOCUS.

The user issues the FIN command to exit FOCUS.

4. Invoke the GENUSQL EXEC.

The user invokes the GENUSQL EXEC.

5. Prepare the Assembler program.

The GENUSQL EXEC invokes the SQLPREP EXEC supplied by IBM on the SQL/DS production disk. SQLPREP precompiles the Assembler program (FEX1 ASMSQL A) and creates a package for it in the SQL/DS database. The package naming convention is 'owner.program_name', where the owner is the COLLID (if one was provided), SQLUID (if one was provided and COLLID was not provided), or the VM userid invoking GENUSQL (if neither COLLID nor SQLUID was provided). In this example, the package is named USER1.FEX1.

The SQLPREP EXEC invokes the IBM preprocessor for Assembler programs (ARIPRPA, residing on the SQL/DS production disk) with the following parameters, provided by GENUSQL: ISOL(USER), BLK, NOPR, DATE(ISO). If the user provides the SQLUID and SQLPSWD parameters, USER=SQLUID/SQLPSWD. Output from the preprocessor is FEX1 LISTPREP A (the preprocessor output listing) and FEX1 ASSEMBLE A (the modified source program). The LISTPREP file contains a summary count of errors and warnings generated by the preprocessor, while the actual text of errors and warnings is in the modified source program.

Successful completion of this step is indicated by the ARI0708I and ARI0796I messages generated by SQL/DS.

6. Assemble the program.

The GENUSQL EXEC assembles the modified source program (FEX1 ASSEMBLE A) using IBM's Assembler H, if available. If it does not find Assembler H, it uses the installed version. This results in object FEX1 TEXT A (a temporary file) and FEX1 LISTING A (the output from the assembly). The Assembler options are LIST and NOALIGN.

Successful completion of this step is indicated by the message displayed at item 6 in the example.

7. Grant the RUN privilege on the package.

GENUSQL issues the CONNECT command to obtain the proper authorization to proceed.

Since GENUSQL was invoked with GRANT=YES, it starts the SQL/DS Database Services Utility program (DBSU). It grants run privileges on USER1.FEX1 to PUBLIC (all users) and issues a COMMIT WORK.

Successful completion of this step is indicated by the ARI0811I and ARI0809I messages generated by SQL/DS.

8. Add the Resource Manager stub.

The GENUSQL EXEC appends the Resource Manager Stub (ARIRVSTC TEXT, residing on the SQL/DS production disk) to FEX1 TEXT A.

A non-zero return code for this step results in an error message.

9. Link-edit the program.

The GENUSQL EXEC links FEX1 TEXT A into STUBLIB LOADLIB A. It then erases FEX1 TEXT A. The linkage editor options are LIST, MAP, XREF, and NCAL.

A non-zero return code for this step results in an error message.

SQL/DS Static MODIFY Example

The following annotated example illustrates the creation of a static MODIFY procedure named MOD1. The numbers 1 through 10 refer to the explanatory notes that follow the example.

Also included for your information are the options FOCUS uses for each step (for example, assembler options). Please refer to the appropriate IBM manual for an explanation of these options and their possible settings.

```
    >
    sql sqlds set static on
    >
1.  compile mod1
    >

    SAMPLE  SQLDS     ON 07/30/91 AT 12.23.48
2.  (FOC1472) STATIC SQL PROGRAM CREATED SUCCESSFULLY
3.  COMPILED...
    >
4.  fin
    Ready; T=0.16/0.27 12:23:54
5.  genusql mod1 (sqluid=admin sqlpswd=minnie grant=yes
    Following installation parameters assumed:
          Pgm     = MOD1
          Sqluid  = ADMIN
          Sqlpswd = MINNIE
          Grant   = YES
          Indvar  = YES
          Collid  =
6.  ARI0717I START SQLPREP EXEC: 07/30/91 12:24:40 EDT
    ARI0320I THE DEFAULT DATABASE NAME IS SQLDBA.
    ARI0663I FILEDEFS IN EFFECT ARE:
    OFFLINE  PRINTER
    SYSIN    DISK     MOD1     ASMSQL   *
    SYSPRINT DISK     MOD1     LISTPREP A1
    SYSPUNCH DISK     MOD1     ASSEMBLE A1
    ARISQLLD DISK     ARISQLLD LOADLIB  W1
    ARI0713I PREPROCESSOR ARIPRPA CALLED WITH THE FOLLOWING PARAMETERS:
          .. PREP=MOD1,USER=ADMIN/********,ISOL(USER),BLK,NOPR,DATE(ISO)
    ARI0708I ALL SQLPREP EXEC PROCESSING COMPLETED SUCCESSFULLY.
    ARI0796I END SQLPREP EXEC: 07/30/91 12:24:45 EDT
```

```
7.  Assembling of MOD1 completed successfully.
8.  ARI0717I START SQLDBSU EXEC: 07/30/91 12:24:52 EDT
    ARI0659I LINE-EDIT SYMBOLS RESET:
             LINEND=# LINEDEL=OFF CHARDEL=OFF ESCAPE=OFF TABCHAR=OFF
    ARI0656I MESSAGE FILE (SYSPRINT): TERMINAL
    ARI0655I INPUT FILE (SYSIN): TERMINAL
    ARI0320I THE DEFAULT DATABASE NAME IS SQLDBA.
    ARI0663I FILEDEFS IN EFFECT ARE:
    ARISQLLD DISK     ARISQLLD LOADLIB  G1
    SYSIN    TERMINAL
    SYSPRINT TERMINAL
    ARI0801I DBS Utility started: 11/07/94 16:21:05.
             AUTOCOMMIT = OFF ERRORMODE = OFF
             ISOLATION LEVEL = REPEATABLE READ
    ARI0828I ...LINEWIDTH reset to 80.
    ARI0827I ...Begin command execution: ERRORMODE = CONTINUE

    ARI0870I Enter the command terminated by semicolon or enter
             EXIT to end.
    ------>
    ------> CONNECT "ADMIN   " IDENTIFIED BY ********;
    ARI8004I User ADMIN connected to server SQLDBA.
    ARI0500I SQL processing was successful.
    ARI0505I SQLCODE = 0 SQLSTATE = 00000 ROWCOUNT = 0

    ARI0870I Enter the command terminated by semicolon or enter
             EXIT to end.
    ------> GRANT RUN ON  ADMIN.MOD1 TO PUBLIC;
    ARI0500I SQL processing was successful.
    ARI0505I SQLCODE = 0 SQLSTATE = 00000 ROWCOUNT = 0

    ARI0870I Enter the command terminated by semicolon or enter
             EXIT to end.
    ------> COMMIT WORK;
    ARI0500I SQL processing was successful.
    ARI0505I SQLCODE = 0 SQLSTATE = 00000 ROWCOUNT = 0

    ARI0870I Enter the command terminated by semicolon or enter
             EXIT to end.
    ------> EXIT
    ARI8997I ...Begin COMMIT processing.
    ARI0811I ...COMMIT of any database changes successful.
    ARI0809I ...No errors occurred during command processing.
    ARI0808I DBS processing completed: 07/30/91 12:24:56.
    ARI0660I LINE-EDIT SYMBOLS RESTORED:
             LINEND=# LINEDEL=¢ CHARDEL=@ ESCAPE=" TABCHAR=ON
    ARI0796I END SQLDBSU EXEC: 07/30/91 12:24:56 EDT
9.  DMSCPY721I Copy ARIRVSTC TEXT W1 APPEND MOD1 TEXT A1 (old file)
    IEW0000     INCLUDE OBJ
    IEW0000     ENTRY STATICSQ
10. IEW0000     NAME MOD1(R)

    Ready; T=0.70/0.93 12:25:05
```

The steps in the process are:

1. Issue the FOCUS COMPILE command.

The user invokes the FOCUS COMPILE facility for MODIFY procedures.

2. Create the ASSEMBLER program.

In this step, FOCUS reads the MODIFY program and creates an ASSEMBLER program with all the embedded SQL statements required by the MODIFY.

The ASSEMBLER program has the same filename as the MODIFY and a filetype of ASMSQL. It is placed on the minidisk accessed as filemode A. In this example, the ASSEMBLER program is called MOD1 ASMSQL A.

3. Create the compiled MODIFY.

FOCUS compiles the MODIFY procedure, creating MOD1 FOCCOMP A.

4. Exit FOCUS.

The user issues the FIN command to exit FOCUS.

5. Invoke the GENUSQL EXEC.

The user invokes the GENUSQL EXEC.

6. Prepare the ASSEMBLER program.

The GENUSQL EXEC invokes the SQLPREP EXEC supplied by IBM on the SQL/DS production disk. SQLPREP precompiles the ASSEMBLER program (MOD1 ASMSQL A) and creates a package for it in the SQL/DS database. The package naming convention is 'owner.program_name', where the owner is the COLLID (if one was provided), SQLUID (if one was provided and COLLID was not provided), or the VM userid invoking GENUSQL (if neither COLLID nor SQLUID was provided). In this example, the package is named ADMIN.MOD1.

The SQLPREP EXEC invokes the IBM preprocessor for ASSEMBLER programs (ARIPRPA, residing on the SQL/DS production disk) with the following parameters, provided by GENUSQL: ISOL(USER), BLK, NOPR, DATE(ISO). If the user provides the SQLUID and SQLPSWD parameters, USER=SQLUID/SQLPSWD. Output from the preprocessor is MOD1 LISTPREP A (the preprocessor output listing) and MOD1 ASSEMBLE A (the modified source program). The LISTPREP file contains a summary count of errors and warnings generated by the preprocessor, while the actual text of errors and warnings is in the modified source program.

Successful completion of this step is indicated by the ARI0708I and ARI0796I messages generated by SQL/DS.

7. Assemble the program.

   The GENUSQL EXEC assembles the modified source program (MOD1 ASSEMBLE A) using IBM’s ASSEMBLER H, if available. If it does not find ASSEMBLER H, it uses the installed version. This results in object MOD1 TEXT A (a temporary file) and MOD1 LISTING A (the output from the assembly). The ASSEMBLER options are LIST and NOALIGN.

   Successful completion of this step is illustrated by the message displayed at item 6 in the example.

8. Grant the RUN privilege on the package.

   GENUSQL issues the CONNECT command to obtain the proper authorization to proceed.

   Since GENUSQL was invoked with GRANT=YES, it starts the SQL/DS Database Services Utility program (DBSU). It grants run privileges on ADMIN.MOD1 to public (all users) and issues a COMMIT WORK.

   Successful completion of this step is indicated by the ARI0811I and ARI0809I messages generated by SQL/DS.

9. Add the Resource Manager stub.

   The GENUSQL EXEC appends the Resource Manager Stub (ARIRVSTC TEXT, residing on the SQL/DS production disk) to MOD1 TEXT A.

   A non-zero return code for this step results in an error message.

10. Link-edit the program.

    The GENUSQL EXEC links MOD1 TEXT A into STUBLIB LOADLIB A. It then erases MOD1 TEXT A. The linkage editor options are LIST, MAP, XREF, and NCAL.

    A non-zero return code for this step results in an error message.

Using the Interface SQL COMPILE and SQL RUN Facilities

The Interface SQL COMPILE facility creates static TABLE procedures. To execute a static TABLE procedure, you must use the Interface SQL RUN facility.

Creating a Static FOCEXEC With the SQL COMPILE Facility

The SQL COMPILE facility offers several options for generating static SQL modules depending on the functions your FOCEXEC performs. Choosing the proper combination of compilation options for each FOCEXEC requires careful consideration.

The syntax for invoking the SQL COMPILE facility is

```
SQL [target_db] COMPILE focexec [-OPTION(options)] [parameters]
END
```

where:

target_db

Indicates the target RDBMS. Acceptable values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

focexec

Is the name of the FOCEXEC to compile.

options

Are execution options, separated by single spaces. No space is allowed between the word -OPTION and the opening parenthesis, and any text between unrecognized option names is ignored. You can use any combination of options listed in the chart that follows the syntax.

parameters

Are pairs of FOCUS ampersand variables with their corresponding values, in the form variable = value. Separate pairs from each other with commas. FOCUS prompts you during the SQL COMPILE for any variables you do not assign in the parameter list (see *SQL COMPILE and SQL RUN Processing* on page 13-37). Depending on where these variables appear in TABLE or MATCH FILE requests, FOCUS either:

- Hard-codes their values into the generated SQL (in which case you must specify the same values for them at run time).
- Generates SQL host variables as placeholders for them (in which case you can specify new values for them at run time).

The SQL COMPILE facility substitutes values for variables that are not part of a TABLE or MATCH FILE request only for the duration of the SQL COMPILE; you can specify new values for them at run time.

The following options are available with the SQL COMPILE facility:

INTERPRET	This option simultaneously executes the FOCEXEC and generates the static SQL module. Use it to compile FOCEXECs containing TABLE requests that execute conditionally depending on output from other TABLE requests (for example, by testing the value of &LINES). Without this option, TABLE requests do not produce any output; therefore, the facility may fail to process some conditionally executed logic. **Note:** The INTERPRET option is mandatory if the FOCEXEC contains MATCH FILE commands.
FROM(fxname)	fxname is the name of a FOCEXEC that calls the FOCEXEC being compiled. No spaces are allowed between the word FROM and the opening parenthesis. Use this option to compile a FOCEXEC that is usually called from another FOCEXEC (for example, a FOCEXEC that displays a menu and sets ampersand variables that are used by the FOCEXEC being compiled).
REPEAT[(n)]	This option processes the FOCEXEC n times, each time appending the generated SQL to the static module being created. No spaces are allowed between the word REPEAT and the opening parenthesis. If you omit n, FOCUS asks after each execution whether to end or execute the FOCEXEC again. Use this option to generate all possible versions of SQL when values assigned to ampersand variables can affect the generated SQL. Assign values for variables by either prompting for them with Dialogue Manager, or by using the TED option.
TED	This option lets you edit the FOCEXEC with the TED editor prior to compiling it. You can use this option in conjunction with the REPEAT option to make incremental changes at each iteration.

Note:

- Any FOCEXEC containing MATCH FILE commands must use the INTERPRET option of the SQL COMPILE command.
- The datatypes of join fields must be equal when accessing multi-segment structures (whether created via the JOIN command or a multi-segment Master and Access File). Also, the scales of packed decimal (P) fields used as join fields must be equal. The scale is the number of digits to the right of the decimal point, specified in the USAGE attribute in the Master File. In addition, the USAGE lengths of any alphanumeric (A) join fields must be equal.
- The SQL COMPILE facility does not generate static SQL statements for HOLD FORMAT DB2 or HOLD FORMAT SQL subcommands in a TABLE request. These subcommands run dynamically, even when the FOCEXEC executes in the SQL RUN environment. The data query (SELECT) portion of the request does, however, generate static SQL statements, if applicable. If necessary, alter the application to generate a flat file and invoke a separate MODIFY that uses static SQL; however, even with this technique, you must issue the CREATE FILE separately, and it must always be dynamic.
- FOCEXECs compiled with the Static SQL COMPILE facility do not support the use of the LOAD command; to execute them, you must use the SQL RUN command.

For more information on the SQL generation process at both SQL COMPILE and SQL RUN time, see *SQL COMPILE and SQL RUN Processing* on page 13-37.

Executing a Static FOCEXEC With the SQL RUN Facility

The syntax for executing a module created by the SQL COMPILE facility is

```
SQL [target_db] RUN focexec [parameters]
END
```

where:

target_db
: Indicates the target RDBMS. Acceptable values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

focexec
: Is the name of a static FOCEXEC module generated by the SQL COMPILE facility.

`parameters`

Are pairs of FOCUS ampersand variables with their corresponding values, in the form variable = value. Separate pairs with commas. If you omit any variables from the parameter list, FOCUS prompts for their values during the SQL RUN execution. Depending on where these variables appear in the TABLE or MATCH FILE requests executed by the FOCEXEC, during the SQL COMPILE FOCUS either:

- Hard-coded their values into the SQL request generated by the SQL COMPILE facility (in which case you must specify the same values for them at run time).
- Generated SQL host variables as placeholders for them (in which case you can specify new values for them at run time).

FOCUS processes variables that are not part of a TABLE or MATCH FILE request in the usual way; you can specify new values for them at run time.

For more information on the SQL generation process at both SQL COMPILE and SQL RUN time, see *SQL COMPILE and SQL RUN Processing* on page 13-37.

SQL COMPILE and SQL RUN Processing

To make optimal use of the Static SQL for TABLE facility, you should understand how the SQL COMPILE and SQL RUN commands generate SQL.

When you invoke the SQL COMPILE facility, TABLE or MATCH FILE commands that access DB2 or SQL/DS tables generate input into two modules:

- A Data Base Request Module (DBRM) for DB2, or ASMSQL file for SQL/DS, containing the static SQL statements to execute at SQL RUN time. Subsequent to the SQL COMPILE process, the DBRM or ASMSQL file is processed by the BIND command (DB2) or GENUSQL EXEC (SQL/DS) to generate an application package in the RDBMS. *The Interface as an RDBMS Application* in Chapter 1, *Introduction*, includes a discussion of bind concepts.
- A STUBLIB load module containing an Assembler program. At SQL RUN time, the STUBLIB load module invokes the static SQL statements from the FOCEXEC. The names of both the DBRM or ASMSQL file and the STUBLIB load module must be identical to the FOCEXEC name. During the SQL COMPILE, multiple RDBMS requests append additional SQL statements to the DBRM or ASMSQL file until the SQL COMPILE facility executes all applicable requests and processes all the resultant SQL statements.

As a result of certain FOCUS TABLE and MATCH FILE request syntax, FOCUS generates SQL host variables in the created SQL statements. Host variables affect both the ability to substitute new variable values at SQL RUN time and RDBMS access path selection. The RDBMS may select different access paths when the request uses host variables because the optimizer must estimate what proportion of records in the table will pass the screening conditions.

In general, the SQL COMPILE facility substitutes an SQL host variable for any literal that appears to the right of a comparison operator in an IF or WHERE screening condition, whether or not the literal represents an ampersand variable in the FOCEXEC. Since FOCUS performs variable substitution prior to invoking the Interface, the Interface cannot distinguish between hard-coded values and values substituted for ampersand variables. Therefore, to support variable substitution for literals in screening conditions at run time, the SQL COMPILE facility assumes that all literals following comparison operators represent variables.

In addition, if varying the values of ampersand variables in a request could change the generated SQL, you must use the REPEAT option of the SQL COMPILE command to enumerate all possible values and generate all possible variations of SQL in a single execution of the SQL COMPILE facility. Examples of such variables include variable filenames or fieldnames in a FOCEXEC, or variables that cause the Interface OPTIMIZATION setting to change, thereby altering the generated SQL.

When you invoke the SQL RUN facility, FOCUS executes the FOCEXEC as usual. In addition, the Interface re-generates SQL statements in memory from any TABLE or MATCH FILE requests that reference DB2 or SQL/DS tables. However, instead of transmitting the SQL to the RDBMS via the dynamic Interface plan, the Interface invokes the STUBLIB load module and loads the RDBMS application plan or package. It then searches the static SQL statements in the plan or package until it finds a statement that exactly matches the re-generated SQL. If the Interface locates an appropriate statement in the plan or package, it executes the statement. If it cannot find a matching statement in the plan or package, it issues a FOC1526 error, and execution terminates.

This run-time regeneration and comparison provides increased flexibility. You can change the source FOCEXEC and, as long as the changes do not affect the generated SQL, you can run the static FOCEXEC without re-submitting it to the SQL COMPILE facility. You need only recompile a FOCEXEC if the generated static SQL statements are different from those generated by the most recent SQL COMPILE.

Note:

- Because of the SQL generation and comparison processing that takes place at SQL RUN time, the Interface OPTIMIZATION setting must be the same at SQL RUN time as at SQL COMPILE time. To ensure that the correct settings are in effect, you should explicitly specify the appropriate SET OPTIMIZATION commands in any FOCEXEC processed, even when using the default setting of ON.

- If a screening condition in a TABLE request tests for the MISSING value, or if you supply the value ‘MISSING’ for an ampersand variable at SQL COMPILE time, the generated SQL contains the NULL value, not a host variable. If the variable might be assigned either the MISSING value or a literal value at SQL RUN time, then you must invoke SQL COMPILE with the REPEAT option to supply the MISSING value in one iteration and another literal place-holder in a separate iteration. This procedure generates two SQL statements, one with the NULL value, and one with a host variable that can accept literal values at SQL RUN time. Without this technique, a FOC1526 may result at SQL RUN time because of the lack of a matching static SQL statement.
- The SQL COMPILE facility does not automatically compile a FOCEXEC called by the currently compiling FOCEXEC. To run a called FOCEXEC statically, you must compile it separately, and invoke it from the calling FOCEXEC with the SQL RUN command at SQL RUN time.
- If an IF or WHERE screening condition uses the ddname option, the file accessed must contain the same number of entries at SQL RUN time as at SQL COMPILE time, or a FOC1526 message results and execution terminates. You may be able to bypass this restriction, where practical, by using the REPEAT option and changing the size of the file with each iteration. Again, however, the number of entries at SQL RUN time must match one of the previously generated SQL statements. This restriction may be lifted in a future release.

  Note: Since the AUTODB2 and AUTOSQL facilities make extensive use of the FOCUS IF/WHERE EQ (ddname) feature, you should not compile them for static SQL.
- If a screening condition in a TABLE request has an ampersand variable to the right of a LIKE operator, different literals used with the same LIKE syntax may produce different SQL. You must ensure that any SQL that could be generated dynamically at SQL RUN time has an exact match in the STUBLIB module; use the REPEAT option of the SQL COMPILE facility, and substitute a different LIKE literal format for each execution. For more information on the SQL generated by the LIKE operator, see Chapter 7, *The Interface Optimizer*. If, at SQL RUN time, the Interface cannot locate a static SQL statement that matches the generated dynamic SQL, it issues a FOC1526 message, and execution terminates.

SQL Host Variable Length Considerations

In order for the RDBMS to consider index access for an SQL predicate containing host variables, the DB2 and SQL/DS query optimizers require the lengths of those host variables to be compatible with the lengths of the corresponding columns in the RDBMS. FOCUS uses the ACTUAL attribute from the Master File to create host variable descriptions in the generated Assembler program. Therefore, the length of the ACTUAL attribute for fields used in screening conditions or as join fields in cross-referenced files (in FOCUS-managed joins) must be equal to the length of the column in the RDBMS.

However, for packed decimal fields, the ACTUAL attribute alone does not provide enough information to determine the exact precision of the RDBMS DECIMAL column. For example, an ACTUAL attribute of P6 could correspond to a DECIMAL column of precision either 10 or 11. The PRECISION attribute of the Access File (described in *PRECISION* in Chapter 4, *Describing Tables to FOCUS*) supplies the required information.

Host Variable Placement

You can activate FSTRACE4 to determine where the Static SQL for TABLE facility inserted host variables in the generated SQL (see Appendix D, *Tracing Interface Processing*, for information on activating Interface traces).

For example, consider the following FEX1 FOCEXEC and SQL COMPILE request (*DB2 Static TABLE Example* on page 13-12 illustrates the compile process for this request):

```
SQL DB2 SET OPTIMIZATION ON
TABLE FILE EMPLOYEE
PRINT EMP_ID SALARY POINTS
WHERE SALARY GE &SALARY
END
```

```
SQL DB2 COMPILE FEX1 SALARY=50000
```

The following command executes the static FEX1 module created by the preceding example:

```
SQL DB2 RUN FEX1 SALARY = 50000
END
```

The FSTRACE4 output from the SQL COMPILE or SQL RUN request follows:

```
SELECT T1.EMP_ID,T1.SALARY,T2.POINTS FROM "USER1"."PAYROLL" T1,
"USER1"."COURSES" T2 WHERE (T2.EMP_NUM = T1.EMP_ID) AND
(T1.SALARY >= :H);
```

The symbol :H (to the right of the comparison operator, >=, in the screening condition on T1.SALARY) represents the SQL host variable. In the example, the SQL RUN command supplies the value 50000 for this variable. Since the variable exists in the generated SQL, you can substitute a value at SQL RUN time, as long as the TABLE request includes a FOCUS ampersand variable in the corresponding location (&SALARY in FEX1).

At SQL COMPILE time, FOCUS substitutes SQL host variables for all literals to the right of logical operators in FOCUS IF and WHERE screening conditions. This substitution allows you to enter values for ampersand variables at SQL RUN time. FOCUS creates these host variables even if the operand is not an ampersand variable. That is, if the literal 50000 had been explicitly coded in the FOCEXEC, FOCUS would still have inserted the :H host variable in the generated SQL. This may have RDBMS optimization implications, especially in the case of SQL range predicates (for example, < or >), because the RDBMS optimizer has to make assumptions concerning data distribution when choosing the access path to the data.

If you are concerned about RDBMS optimization in the DB2 environment, execute the SQL COMPILE command with the STATIC option set to NOBIND, and execute a separate BIND using the EXPLAIN option. (For more information about BIND options, consult the IBM *DB2 Command and Utility Reference*.) In the SQL/DS environment, you can use the EXPLAIN option of the GENUSQL EXEC. (For more information about this option, consult the explanation of the SQLPREP command in the IBM *SQL/DS Application Programming for VM Systems* manual.)

In many cases, you can use the FOCUS DEFINE facility to insert literal values, rather than host variables, into the generated SQL. For example, suppose the FEX1 FOCEXEC had the value 50000 hard coded into the screening condition. FOCUS would still substitute the host variable, :H, at SQL COMPILE time. The DB2 optimizer, unaware of the 50000 value at BIND time, might select a tablespace scan access path. This type of scan is inefficient if the SALARY column is indexed, because DB2 will only need to retrieve a few rows when the value 50000 is substituted for :H at SQL RUN time.

You can alter the request as follows:

```
SQL DB2 SET OPTIMIZATION ON
DEFINE FILE EMPLOYEE
SALVALUE/P13.2 = 50000;
TABLE FILE EMPLOYEE
PRINT EMP_ID SALARY POINTS
WHERE SALARY GE SALVALUE
END
```

The generated SQL now includes the literal value rather than the host variable:

```
SELECT T1.EMP_ID,T1.SALARY,T2.POINTS FROM "USER1"."PAYROLL" T1,
"USER1"."COURSES" T2 WHERE (T2.EMP_NUM = T1.EMP_ID) AND
(T1.SALARY >= 50000);
```

The DB2 optimizer can now choose a direct index access path to the data because it knows the literal screening value at BIND time.

This solution is not valid for literals in LIKE predicates, as FOCUS does not support field names to the right of the LIKE operator in TABLE requests. Also, this technique generates dual range predicates for FOCUS WHERE/FROM-TO clauses, not an SQL BETWEEN. The use of dual range predicates may have optimization implications.

Note: If a screening condition in a TABLE request has an ampersand variable to the right of a LIKE operator, different literals used with the same LIKE syntax may produce different SQL. You must ensure that any SQL that could be generated dynamically at SQL RUN time has an exact match in the STUBLIB module. You can use the REPEAT option of the SQL COMPILE facility and substitute a different LIKE literal format for each iteration. For more information on the SQL generated by the LIKE operator, see Chapter 7, *The Interface Optimizer*. If, at SQL RUN time, the Interface cannot locate a static SQL statement that matches the generated dynamic SQL, it issues a FOC1526 message and terminates execution.

Resource Restrictions

Static compilation creates an Assembler routine that includes SQL statements. Each SQL statement uses a certain number of Assembler resource units (base registers) from the finite number of base registers available. Exceeding this limit might, in some cases, prohibit a large FOCEXEC from compiling successfully. These cases, however, should be extremely rare, and should only involve extremely large generated SQL SELECT statements. The vast majority of applications will be unaffected. These limitations are not unique to FOCUS; they apply to any Assembler program with embedded SQL.

The Interface uses an optimization algorithm to allocate resources from the available pool of 16 registers. If it determines that a limit has been exceeded, it issues an error message during static compilation. Therefore, if your FOCEXEC already exists, the easiest way to determine if it exceeds any limits is to statically compile it.

If the Interface determines that a base register limit has been exceeded during the creation of the Assembler program, it issues the FOC1355 and FOC1359 error messages and terminates compilation. In some cases, the Interface cannot determine whether limits have been exceeded until the program is actually assembled. It flags these errors as addressability errors during the assembly and adds them to the Assembler listing file that it generates.

14 CALLDB2: Invoking Subroutines Containing Embedded SQL

Topics:

- Implementation
- Creating CALLDB2-Invoked Subroutines
- Run-time Requirements
- Sample Dialogue Manager Procedure for Invoking CALLDB2

CALLDB2, a FOCUS subroutine included in the FOCUS DB2 Interface, provides a standard method for invoking user-written subroutines that make embedded SQL calls to DB2.

Prior to CALLDB2, you had to bind subroutines containing embedded SQL with the application plan for the Interface, so they generated increased administrative overhead. The applications programmer was responsible for run-time plan management (making sure to invoke the correct application plan with the Interface SET PLAN command), and for instructing the Interface to establish a thread to DB2 before invoking the subroutine.

CALLDB2 eliminates the requirement to include Interface Database Request Modules (DBRMs) in the application plan for the subroutine. In addition, CALLDB2 ensures that subroutines do not disrupt the current DB2 Interface thread. The Interface automatically closes any existing thread to DB2 and opens a new thread to the application plan for the subroutine. After the subroutine completes, the Interface restores the original environment. With CALLDB2, you can invoke subroutines when no prior thread to DB2 exists.

Note: The CALLDB2 feature requires that the DB2 Interface be installed to use the Call Attachment Facility (CAF). You can verify this by issuing the SQL DB2 ? command; Call Attach should be ON.

For additional documentation, consult the *FOCUS for IBM Mainframe User Written Subroutines Library*. Read it before writing subroutines that use CALLDB2. **Note:** Any additions, differences, or limitations described in this chapter override that document.

Implementation

CALLDB2 is an Information Builders-supplied subroutine you invoke from a Dialogue Manager program. The syntax is

```
-SET &var = CALLDB2('subrtine','planname',input1,input2,...,['format']);
```

where:

`&var`

Is the Dialogue Manager variable to receive the value returned by the subroutine.

`subrtine`

Is a subroutine that uses embedded SQL. The subroutine name can be up to eight characters long (unless you are writing the subroutine in a language that allows less) and must be enclosed in single quotation marks. The first character must be a letter (A-Z); each additional character can be a letter or a number. You must pad the name on the right with blanks if it is less than eight characters long.

`planname`

Is the DB2 application plan for your subroutine. The plan name can be up to eight characters long and must be enclosed in single quotation marks. You must pad the name on the right with blanks if it is less than eight characters long.

`input1...`

Are the input arguments. You must know what arguments the subroutine requires, their formats, and the order in which to specify them. You can include up to 26 input arguments (normally, user written subroutines allow 28, but CALLDB2 uses two of them for subroutine name and plan name).

`format`

Is the format of the output value, enclosed in single quotation marks. This parameter is optional. If you do not provide a format, the default format is determined by the format of the last element in the calling list. If your program does not return an output value, the default return value is the last element in the calling list. In cases of error, the returned value is '*ERROR* '.

Although Dialogue Manager variables contain only alphanumeric data, they can serve as numeric arguments; the -SET command converts their alphanumeric values to double-precision format before passing them to the subroutine. However, if a subroutine returns a numeric value and you set a Dialogue Manager variable to this value, FOCUS truncates the output to an integer and converts it to a character string before storing it in the variable.

Note: You can invoke CALLDB2 only from Dialogue Manager -SET, -IF, or -TSO RUN control statements. You cannot invoke it from DEFINE, COMPUTE, MODIFY VALIDATE or IF statements, or from Extended Matrix Reporting (also known as Financial Reporting Language) RECAP statements.

Creating CALLDB2-Invoked Subroutines

The steps for creating a CALLDB2-invoked subroutine are:

1. Write the subroutine program logic.
2. Precompile the subroutine.
3. Compile or assemble the subroutine.
4. Link-edit the subroutine.
5. BIND the subroutine.

The Interface as an RDBMS Application in Chapter 1, *Introduction*, includes a discussion of bind concepts.

Write the Subroutine

Your subroutine should assume that the Interface will do all thread handling. This includes connection to DB2, disconnection from DB2, and the opening and closing of threads. In addition, do not code COMMIT WORK or ROLLBACK WORK statements within the subroutine itself. The Interface tracks open logical units of work (LUWs), and it cannot detect a COMMIT WORK or ROLLBACK WORK in a subroutine.

Since CALLDB2 is an Interface command, its behavior is affected by the Interface SET AUTO*action* ON *event* command (see Chapter 10, *Thread Control Commands*, for more information about this command). If AUTOCOMMIT is set ON COMMAND, the default setting, the Interface assumes there is an open LUW and automatically issues a COMMIT WORK at the end of a CALLDB2 subroutine.

If your subroutine requires the Interface to issue a conditional COMMIT or ROLLBACK WORK:

1. Set AUTOCOMMIT on FIN immediately before entering the subroutine to prevent the Interface from generating an automatic COMMIT WORK at the end of the CALLDB2 command.
2. Exit the subroutine and return a value to the Dialogue Manager procedure indicating which action, COMMIT or ROLLBACK, the procedure should take on behalf of the subroutine.
3. After your Dialogue Manager procedure issues the COMMIT or ROLLBACK WORK command, reset AUTOCOMMIT on COMMAND to restore the default Interface environment.

The example in *Sample Dialogue Manager Procedure for Invoking CALLDB2* on page 14-9 illustrates this technique.

A CALLDB2 subroutine is also subject to the rules of Interface plan management. Two types of plan management are defined for the Interface: basic and extended. If you use basic plan management, the Interface automatically switches plans as required. The alternative, extended plan management, is invoked by the Interface SET PLAN command. In this case, the application program, not the Interface, manages plans. The discussion in *Plan Management in DB2* in Chapter 13, *Static SQL*, applies to CALLDB2 subroutines.

The following very simple COBOL program illustrates the steps involved in preparing a static subroutine for use with CALLDB2:

```
    ID DIVISION.
    PROGRAM-ID. TESTDB2.
    ENVIRONMENT DIVISION.
    DATA DIVISION.
    WORKING-STORAGE SECTION.
        EXEC SQL INCLUDE SQLCA END-EXEC.
    LINKAGE SECTION.
    01  VALUE1                 PIC X(8).
    01  VALUE2                 PIC X(8).
    01  RETCODE                PIC S9(9) COMP SYNC.
1.  PROCEDURE DIVISION USING VALUE1, VALUE2, RETCODE.
2.      EXEC SQL WHENEVER SQLERROR GO TO PROGRAM-EXIT
        END-EXEC.
3.      EXEC SQL INSERT INTO USER1.TESTDB2
           (COL1)
           VALUES(:VALUE1)
        END-EXEC.
        EXEC SQL INSERT INTO USER1.TESTDB2
           (COL1)
           VALUES(:VALUE2)
        END-EXEC.
4.  PROGRAM-EXIT.
        MOVE SQLCODE TO RETCODE.
        GOBACK.
    //**
```

The subroutine:

1. Accepts two input values and returns a value named RETCODE.
2. Checks the SQLCODE after each SQL command. If it encounters a negative SQLCODE, the subroutine branches to PROGRAM-EXIT.
3. Issues two SQL INSERT commands that place the input values into a single-column DB2 table.
4. Passes the value of the SQLCODE back to FOCUS. If the procedure encountered an error, it returns the negative SQLCODE from that error back to the Dialogue Manager routine. Otherwise, it passes the last SQLCODE, which should be zero, back to the Dialogue Manager routine. The Dialogue Manager routine will check the value of the returned SQLCODE and issue an SQL COMMIT WORK or ROLLBACK WORK depending on the returned value.

The following steps prepare the sample subroutine, TESTDB2, for execution. JCL for each step is provided for illustrative purposes. In general, these steps apply to any user-written subroutine containing embedded SQL. Your JCL will vary depending on such factors as choice of language and site-specific requirements or conventions. Consult the appropriate IBM manual for help with each step.

Precompile the Subroutine

This example includes the sample program as in-stream input to the precompilation job step:

```
//Job card goes here...
//*****************************************************************
//*** PRECOMPILE COBOL II CODE CONTAINING EMBEDDED (STATIC) SQL ***
//*****************************************************************
//PC       EXEC PGM=DSNHPC,
//          PARM='HOST(COB2),OPTIONS,SOURCE,XREF'
//STEPLIB  DD DISP=SHR,DSN=DSN510.SDSNLOAD
//DBRMLIB  DD DISP=SHR,DSN=prefix.DBRMLIB.DATA(TESTDB2)
//SYSCIN   DD DSN=prefix.PRECOMP.OUTPUT,DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,SPACE=(TRK,(20,10),RLSE),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=4080)
//SYSPRINT DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//SYSUT1   DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT2   DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT3   DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT4   DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSUT5   DD UNIT=SYSDA,SPACE=(800,(500,500),,,ROUND)
//SYSIN    DD *
ID DIVISION.
  PROGRAM-ID. TESTDB2.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
    EXEC SQL INCLUDE SQLCA END-EXEC.
LINKAGE SECTION.
01  VALUE1                 PIC X(8).
01  VALUE2                 PIC X(8).
01  RETCODE                PIC S9(9) COMP SYNC.
PROCEDURE DIVISION USING VALUE1, VALUE2, RETCODE.
    EXEC SQL WHENEVER SQLERROR GO TO PROGRAM-EXIT
    END-EXEC.
    EXEC SQL INSERT INTO USER1.TESTDB2
         (COL1)
         VALUES(:VALUE1)
    END-EXEC.
    EXEC SQL INSERT INTO USER1.TESTDB2
         (COL1)
         VALUES(:VALUE2)
    END-EXEC.
PROGRAM-EXIT.
    MOVE SQLCODE TO RETCODE.
    GOBACK.
//**
```

The precompilation process produces a modified source program (the data set allocated to DDNAME SYSCIN) and a database request module or DBRM (member TESTDB2 in the data set allocated to DDNAME DBRMLIB).

Compile or Assemble the Subroutine

After precompilation, compile or assemble the modified source program:

```
//Job card goes here...
//***************************************
//*** COBOL II COMPILE OF SOURCE CODE ***
//***************************************
//COB2     EXEC PGM=IGYCRCTL,PARM='OBJECT',REGION=1024K
//STEPLIB  DD DSN=VSCOBOL2.V1R3M2.COB2COMP,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSLIN   DD DSN=prefix.OBJECT.MODULE,DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,SPACE=(TRK,(20,10),RLSE)
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT6   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSIN    DD DSN=prefix.PRECOMP.OUTPUT,DISP=(OLD,DELETE)
//**
```

The output of the compilation in this example is the data set allocated to DDNAME SYSLIN.

Link-Edit the Subroutine

Link-edit the output of the compilation or assembly to produce a run-time module:

```
//Job card goes here...
//**********************************
//*** LINK EDIT OBJECT MODULE(S) ***
//**********************************
//LKED     EXEC PGM=IEWL,PARM='LIST,XREF,LET,MAP',REGION=512K
//SYSLIN   DD DSN=prefix.OBJECT.MODULE,DISP=(OLD,DELETE)
//         DD *
 INCLUDE DB2(DSNALI)
 INCLUDE DB2(DSNHADDR)
 MODE AMODE(31),RMODE(ANY)
 NAME TESTDB2(R)
/*
//SYSLMOD  DD DSN=prefix.TESTDB2.LOAD,DISP=SHR
//SYSLIB   DD DSN=VSCOBOL2.V1R3M2.COB2LIB,DISP=SHR
//DB2      DD DSN=DSN510.SDSNLOAD,DISP=SHR
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPRINT DD SYSOUT=*
//
/*
```

The run-time module in this example is the data set allocated to DDNAME SYSLMOD. You must include DSNALI, the language interface, during link-edit. DSNALI is in the DB2 load library (DSN510.SDSNLOAD in the example). The link-edit of the sample program also includes DSNHADDR, used to move values to and from the SQL Descriptor Area (SQLDA). DSNHADDR is required only for COBOL programs.

BIND the Subroutine

Now, bind the DBRM created during precompilation into an application plan, and store it in the DB2 database:

```
//Job card goes here...
//**********************************
//*** BIND THE PLAN              ***
//**********************************
//BIND EXEC   PGM=IKJEFT01,DYNAMNBR=10
//STEPLIB   DD DSN=DSN510.SDSNLOAD,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSTSPRT  DD SYSOUT=*
//DBRMLIB   DD DSN=prefix.DBRMLIB.DATA,DISP=SHR
//SYSTSIN   DD *
DSN SYSTEM(DB2P)
BIND PLAN(TESTDB2) MEMBER(TESTDB2) -
                   ACTION(REPLACE) ISOLATION(RR)
END
//*
```

The BIND subcommand creates an application plan called TESTDB2 that contains SQL statements from the DBRM named TESTDB2.

Refer to the *DB2 Command and Utility Reference* for an explanation of possible BIND parameters.

Note: If you make any changes to your subroutine, you have to repeat the precompilation, compilation or assembly, link-edit, and BIND steps (see *Precompile the Subroutine* on page 14-6, through *BIND the Subroutine* on page 14-8).

Run-time Requirements

Add the run-time library for your subroutine ('prefix.TESTDB2.LOAD' in the example) to either the CLIST allocation for USERLIB or the JCL allocation for STEPLIB (making sure the data set with the largest blocksize appears first). For example

```
ALLOC F(USERLIB)  DA('prefix.TESTDB2.LOAD' -
                     'prefix.FOCSQL.LOAD'  -
                     'prefix.FOCLIB.LOAD'  -
                     'prefix.FUSELIB.LOAD') SHR REU
```

or:

```
//STEPLIB   DD DSN=prefix.TESTDB2.LOAD,DISP=SHR
//          DD DSN=prefix.FOCSQL.LOAD,DISP=SHR
//          DD DSN=prefix.FOCLIB.LOAD,DISP=SHR
//          DD DSN=prefix.FUSELIB.LOAD,DISP=SHR
```

Sample Dialogue Manager Procedure for Invoking CALLDB2

This sample Dialogue Manager procedure invokes the TESTDB2 subroutine:

```
1. SQL DB2 SET AUTOCOMMIT ON FIN
   -RUN
   -*
2. -SET &RUNOK = CALLDB2('TESTDB2 ','TESTDB2 ','BLUE','RED','I4') ;
   -RUN

3. -SET &LUW = IF &RUNOK NE 0 THEN 'SQL DB2 ROLLBACK WORK' ELSE
   -           'SQL DB2 COMMIT WORK' ;
4. &LUW
   -RUN

5. SQL DB2 SET AUTOCOMMIT ON COMMAND
   -RUN
```

The procedure:

1. Sets AUTOCOMMIT on FIN to prevent the Interface from issuing an automatic COMMIT WORK at the end of the subroutine.
2. Invokes CALLDB2 for subroutine TESTDB2. The return code from the subroutine will be stored in Dialogue Manager variable &RUNOK.
3. Tests the value of the return code and, depending on the result of the test, stores either the SQL COMMIT WORK or the SQL ROLLBACK WORK command in Dialogue Manager variable &LUW.
4. Issues the COMMIT or ROLLBACK command.
5. Sets AUTOCOMMIT on COMMAND to restore Interface default behavior.

A Additional Topics

Topics:

Status Return Variable: &RETCODE

The Dialogue Manager status return variable, &RETCODE, indicates the status of FOCUS query commands. You can use it to test RDBMS return codes. The &RETCODE variable contains the last return code resulting from an executed report request, MODIFY request, or native SQL command (issued with or without Direct SQL Passthru). See Appendix E, *SQL Codes and Interface Messages*, for a list of common SQL return codes.

In a Dialogue Manager request, you can use a -IF statement to test the &RETCODE value against a specified SQL return code. Then, you can take corrective actions based on the result of the -IF test. An SQL return code of zero (0) indicates a successful execution, a positive return code indicates a warning, and a negative return code indicates an error.

For example, consider the following FOCEXEC. It issues the FOCUS CREATE FILE command to create the EMPINFO table. The FOCEXEC tests for an SQL return code of -601, generated when a table already exists. The -TYPE statement displays message text. The first -TYPE statement displays the &RETCODE value; the second explains that the table exists.

```
CREATE FILE EMPINFO
-RUN
-TYPE RETCODE IS  &RETCODE
-IF &RETCODE EQ -601 GOTO DUPL;
-EXIT
-DUPL
-TYPE  THIS TABLE ALREADY EXISTS. SPECIFY ANOTHER TABLE NAME.
-EXIT
```

When you execute the FOCEXEC, the following messages display:

```
>  (FOC1400) SQLCODE IS  : -601/FFFFFDA7
(FOC1421) TABLE EXISTS ALREADY. DROP IT OR USE ANOTHER TABLENAME
(FOC1414) EXECUTE IMMEDIATE ERROR.
RETCODE IS     -601
THIS TABLE ALREADY EXISTS. SPECIFY ANOTHER TABLE NAME
>
```

Since the table already exists, the RDBMS generates the SQL return code -601, and the &RETCODE variable stores the code. The expression in the -IF statement is true, and the text messages display.

Note: Another useful Dialogue Manager variable, &FOCERRNUM, stores the last FOCUS or Interface (not RDBMS) error number generated by the execution of a FOCEXEC. See the *FOCUS for IBM Mainframe User's Manual* for information about &FOCERRNUM and other statistical variables.

Default DATE Considerations

Starting with FOCUS Version 6.8, the default DATE value the Interface uses for the RDBMS DATE datatype changed. If you use the FOCUS MODIFY facility to maintain RDBMS tables containing DATE columns, this change may have some impact on your applications. If you have a read-only version of the Interface, or if your site does not use MODIFY, you will not be affected.

You can control the default DATE value with the Interface SET DEFDATE command.

The Default DATE Value

The Interface uses the default DATE value in conjunction with RDBMS DATE columns described in a Master File as ACTUAL=DATE. Under certain circumstances, if your MODIFY procedure does not provide a value for a DATE column, the Interface substitutes the default value. In FOCUS Version 6.8 and up, the default value has changed to make Interface DATE behavior more closely resemble that of the FOCUS DBMS.

In prior releases of FOCUS, the Interface default value for RDBMS tables was '1901-01-01' (for the sake of convenience, all DATE values are in DB2 ISO format unless otherwise indicated). The FOCUS DBMS default (or base) value has always been '1900-12-31'.

With the FOCUS DBMS, base DATE values print as blanks in report output by default. (This discussion assumes that the FOCUS DATEDISPLAY parameter is OFF, the default. The SET DATEDISPLAY = ON command displays the base date in FOCUS reports. See your FOCUS documentation for details.) The old Interface default DATE value always displayed on reports.

When DEFDATE is NEW, the Interface base DATE value is identical to the FOCUS DBMS base date: 1900-12-31, and it displays the same way in reports.

The Interface SET DEFDATE Command

You can control the Interface default date with the following Interface SET command

```
SQL [target_db] SET DEFDATE {NEW|OLD}
```

where:

target_db

Indicates the target RDBMS. Valid values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

OLD

The Interface supplies the old default date, '1901-01-01'. OLD is the default in FOCUS Version 6.5.

NEW

The Interface supplies the new default date, '1900-12-31'. NEW is the default starting in FOCUS Version 6.8.

Effects on Existing Applications

With the new default date, TABLE requests no longer require DEFINE or COMPUTE statements in order to display blanks instead of default dates. Applications that test for default values require updating to reflect the new default date.

For the sake of consistency, you may wish to update your databases to change old default values to the new default. You can use the following code to change old default values

```
SQL [target_db] UPDATE creator.tablename
    SET date_column = '1900-12-31' WHERE date_column = '1901-01-01'
END
```

where:

target_db

Is DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

Note: The sample code uses Direct SQL Passthru syntax. You can also issue this code through the Interface native SQL command syntax without invoking Direct SQL Passthru.

The change in the default DATE value affects only the selection of default values supplied by MODIFY procedures under certain circumstances, and whether default values appear on a FOCUS report. In all other respects (for example, in screening conditions), the default value is a valid DATE value. If you use a report writer other than FOCUS, any default DATE values in the RDBMS table will display as either '1901-01-01' or '1900-12-31', depending on the FOCUS release that placed them in the table and the value of the SET DEFDATE command.

Chart: FOCUS DATE Values for User Input Values

The following chart summarizes how FOCUS stores DATE values in response to specific user input values for non-conditional and conditional data entry in MODIFY:

				If DEFDATE is NEW:	
MISS -ING	**User Input**	**FOCUS DBMS Report Output**	**OLD RDBMS Report Output**	**RDBMS Report Output**	**Value Stored in RDBMS**
Non-conditional data entry (<date_column>) in MODIFY					
ON	blank	blank	1901/01/01	blank	1900-12-31
ON	'.'	NODATA	NODATA	NODATA	null
ON	0 (zero)	blank	1901/01/01	blank	1900-12-31
OFF	blank	blank	1901/01/01	blank	1900-12-31
OFF	'.'	input value rejected by MODIFY			
OFF	0 (zero)	blank	1901/01/01	blank	1900-12-31
Conditional data entry (<date_column) in MODIFY					
ON	blank	NODATA	NODATA	NODATA	null
ON	'.'	NODATA	NODATA	NODATA	null
ON	0 (zero)	blank	1901/01/01	blank	1900-12-31
OFF	blank	blank	1901/01/01	blank	1900-12-31
OFF	'.'	input value rejected by MODIFY			
OFF	0 (zero)	blank	1901/01/01	blank	1900-12-31

Note:

- FOCUS displays the NODATA value whenever a column contains a null value. The default NODATA value is the period ('.').
- Column 1 shows the MISSING parameter in the Master File for the column described as ACTUAL=DATE.
- Column 2 shows the value the MODIFY user enters for that column.
- Column 3 shows, for the FOCUS DBMS, how that value would display in FOCUS report requests. If a blank appears on the report, the stored value is '1900-12-31'.
- Column 4 shows, for an RDBMS table, how the entered value displays in a FOCUS report if DEFDATE is OLD. You can see the value FOCUS stored, since it appears on the report instead of a blank.

- Column 5 shows, again for an RDBMS table, how the entered value displays on a FOCUS report provided DEFDATE is NEW. The report output is now identical to the report produced for the FOCUS DBMS (Column 3).
- Column 6 shows the actual default value stored in the RDBMS table for values entered when DEFDATE is NEW.

The chart shows how values are stored and displayed for conditional (single caret) and unconditional (double caret) fields.

Note: If you enter a blank for a conditional field, and if a value already exists for that field, the field is not updated. The chart shows what happens when you enter blanks for conditional fields that have no prior values.

If you want to be sure that FOCUS does not store a default value regardless of the user’s input, have the application program check the entered value. If the user enters 0 (zero) or blank, COMPUTE the DATE field as 'MISSING' to make FOCUS set the column to NULL in UPDATE or INSERT statements. This technique works only if the DATE column allows nulls and is described to FOCUS as MISSING=ON.

See the *FOCUS for IBM Mainframe User’s Manual* if you are not familiar with the terms in the preceding discussion.

Standard FOCUS and Interface Differences

In the design of the Interface, every effort has been made to retain compatibility with the FOCUS mainframe product. Concepts embodied in the *FOCUS for IBM Mainframe User’s Manual* become portable across environments. In some situations, however, these goals conflict with the need for pragmatic and efficient use of the relational model as implemented within the RDBMS.

In the following areas, results may be different from those you expect, or some features may not be available:

- When describing an embedded join in a multi-table Master File, the FOCUS FIELDTYPE keyword (FIELDTYPE=I) is not required for the cross-referencing field of the SQL table. The Interface ignores the keyword.
- You may not use the logical relations INCLUDES and EXCLUDES with non-FOCUS files in a FOCUS IF or WHERE test.
- FOCUS GROUP fields are not supported.
- The FOCUS CREATE FILE command cannot write over an existing table as it can with a FOCUS database. Instead, you must use the RDBMS DROP TABLE command to drop the table before you can recreate it.
- HOLD FORMAT SQL tables created with the default name HOLD are not dropped at the end of a FOCUS session. To drop the table, issue the RDBMS DROP TABLE command.

- JOIN results (embedded or dynamic) are a function of Master Files or a JOIN specification and an OPTIMIZATION setting. All variations are discussed in Chapter 8, *Advanced Reporting Techniques*.
- Since MODIFY FILE does not create a table if it does not exist, you must create the table prior to executing the MODIFY.
- With the FOCUS Direct SQL Passthru facility, you can invoke all native SQL statements, including SELECT statements, from within FOCUS. Without Direct SQL Passthru, you can invoke most native SQL statements from within FOCUS, but not those that return data in addition to return codes. SELECT is an example of an SQL statement that returns data and that, therefore, must be executed with Direct SQL Passthru.
- The maximum number of bytes per native SQL request is 4096 for requests issued without Direct SQL Passthru. With Direct SQL Passthru, the maximum is 32764 bytes.
- A COMBINE statement in a MODIFY procedure may link several external (non-FOCUS DBMS) files. However, you cannot use the COMBINE command to link a FOCUS database with an RDBMS table.
- The Interface allows you to change the value of a table’s primary key with the UPDATE command in a MODIFY MATCH statement. When modifying a FOCUS database, you cannot change key field values. Also, with the Interface, you can match on any field or combination of fields in a row; you need not include the full primary key.
- MODIFY does not support FOCUS alternate file views and remote segment descriptions.
- When modifying non-FOCUS files, including RDBMS tables, you must code the keywords TRACE or ECHO on the line following the MODIFY FILE statement in order for tracing or echoing to occur.

Differences Between the DB2 and SQL/DS Interfaces

There are some minor differences between the DB2 and SQL/DS Interfaces as a result of their operating environments. The following list outlines these differences:

- To use the DB2 Interface, you must have access to partitioned data sets allocated to the following DDNAMEs: MASTER, FOCSQL, and FOCEXEC. Your DBA must authorize you to execute the FOCUS PLAN used at your site.
- AUTODB2 is the procedure that automatically generates the FOCUS Master and Access Files in the DB2 Interface. The SQL/DS Interface uses AUTOSQL.
- EXPDB2 (or EXPDB231) is the procedure that runs the FOCUS EXPLAIN utility in the DB2 Interface. The SQL/DS Interface uses EXPSQL (or EXPSQL34). See Chapter 7, *The Interface Optimizer*, for information about the FOCUS EXPLAIN utility.

- The AUTODB2 procedure uses the system catalog tables SYSCOLUMNS, SYSINDEXES, SYSKEYS, and SYSTABLES. AUTOSQL uses SYSCOLUMNS and SYSINDEXES.
- The AUTODB2 procedure requires the following allocations:
    - AUTODB2 must be a member in the FOCEXEC PDS.
    - DB2CAT, DB2SYSTB, and AUTODB2 must be members in the MASTER PDS.
    - DB2CAT and DB2SYSTB must be members in the FOCSQL PDS.
- SYSIBM is the creator of all system catalog tables in a DB2 environment. The SQL/DS environment uses SYSTEM.
- The DBSPACE default for DB2 tables is database DSNDB04. For SQL/DS the default is the user's private dbspace.
- The default Isolation Level for static MODIFY procedures in SQL/DS is CS. To invoke the Repeatable Read Isolation Level (RR), the MODIFY itself must contain an SQL SET ISOLATION RR command.

  In DB2, the default Isolation Level for static MODIFY procedures is RR. To change this default, you must leave the FOCUS environment and issue a separate DB2 BIND with an Isolation Level of CS. Consult the *IBM DB2 Command and Utility Reference* for additional information on the BIND command.
- The DB2 CURRENT SQLID security facility is not available in SQL/DS.
- The SQL/DS CONNECT security facility is not available in DB2.

Long Fieldname Considerations

In Version 6.8, many limitations on long fieldnames and aliases were lifted. For a complete discussion of long and qualified fieldname support, consult the *FOCUS for IBM Mainframe User's Manual*.

Note: If a request written for a prior release of FOCUS contains truncated fieldnames, and if you rerun AUTODB2 or AUTOSQL on the table, the AUTO facility may generate fieldnames in the new Master File that are different from the names it generated before long fieldnames were supported.

Limitations on Long Fieldnames

The following limitations apply to long fieldnames and aliases:

- In a Master File, the FIELDNAME and ALIAS attributes may not be qualified.
- Long fieldnames are truncated to 12 characters in CHECK FILE PICTURE operations.

- In ModifyTalk, fieldname, title, and field length may not exceed 80 characters.

Querying the RDBMS SYSCOLUMNS Catalog

With proper authorization, you can read the RDBMS catalog table, SYSCOLUMNS, as an alternate source of column information. For easy access to the catalog table, the Interface provides Master Files DB2CAT for DB2 and SYSCOL for SQL/DS. You can issue a FOCUS report request that queries the catalog table for column characteristics, creator ids, and table names.

For example, this FOCUS request lists column names, datatypes, and lengths for each table created by USER1:

```
table file syscol
print
?f
    CNAME        TNAME         CREATOR        COLNO           COLTYPE
    LENGTH       SYSLENGTH     NULLS          REMARKS         COLCOUNT
    HIGH2KEY     LOW2KEY       AVGCOLLEN      ORDERFIELD

cname coltype length
by tname
where creator is 'user1'
end

NUMBER OF RECORDS IN TABLE=       84  LINES=     84
```

The first page of the report follows:

```
TNAME        CNAME             COLTYPE                LENGTH
-----        -----             -------                ------
ADDRESS      EID               CHAR                   9
             LN1               CHAR                   20
             AT                CHAR                   4
             LN2               CHAR                   20
             LN3               CHAR                   20
             ANO               INTEGER
BILLING      PART_NUM          CHAR                   7
             ORDER_NO          INTEGER
             DELIVER_DT        DATE
             INVOICE_DT        DATE
             INVOICE_AMT       DECIMAL                (15,3)
             DEL_INVOICE       INTEGER
CLIENTS      C_STREET          CHAR                   20
             C_CITY            CHAR                   10
             CUSTOMER_NUM      CHAR                   4
             C_ZIP             CHAR                   5
             C_COUNTRY         CHAR                   6
             BRANCH_NAME       CHAR                   8
```

Determining Decimal Notation at Run-time

The DB2 Interface generates the appropriate SQL for whichever decimal point notation was selected (period or comma) when the IBM DB2 software was installed, regardless of the FOCUS Continental Decimal Notation parameter setting. You can then set the Continental Decimal Notation parameter to control how FOCUS displays the numbers.

Continental Decimal Notation uses a comma to mark the decimal position in a number and uses periods (.) for separating significant digits into groups of three. In FOCUS, to set the Continental Decimal Notation parameter, issue the following

```
SET CDN= {OFF|ON}
```

where:

`OFF`
: Turns off Continental Decimal Notation. OFF is the default.

`ON`
: Turns on Continental Decimal Notation.

Regardless of the CDN setting, the Interface generates the appropriate SQL for the decimal point notation (comma or period) selected when the IBM DB2 software was installed using the supplied IBM Application Programming Defaults Panel DSNTIPF. For more information on setting IBM DB2 application defaults, refer to the *IBM DB2 Administration Guide*. The SET CDN command controls how numbers are displayed in the FOCUS environment.

For example, given the number 3,045,000.76:

`SET CDN = ON`	Displays the number as 3.045.000,76.
`SET CDN = OFF`	Displays the number as 3,045,000.76.

Note: In order for this feature to function correctly, the proper DB2 DSNEXIT data set (for example, DSN510.SDSNEXIT) must be first in the sequence of data sets allocated to DDNAME STEPLIB. If this is not possible at your site, you can place the DSNEXIT data set in the FOCLIB concatenation instead, as FOCUS searches FOCLIB before STEPLIB.

Remote Segment Descriptions

Remote segment descriptions simplify the process of describing hierarchies of RDBMS tables to FOCUS. You can use them for Master and Access Files that provide read-only access to RDBMS tables; you cannot use the same descriptions for MODIFY procedures.

The Interface allows you to create multi-table Master and Access Files that define RDBMS tables or views as segments in the description (see Chapter 5, *Multi-Table Structures*). Each Master File segment description consists of a segment declaration followed by descriptions of all of the fields in the segment (columns of the corresponding table).

It is not unusual for several Master Files to contain a segment description for the same RDBMS table. If a table's description is detailed in one Master File, you can automatically incorporate that description in other Master Files. The syntax is

```
SEGNAME=segname, PARENT=parent, SEGTYPE= {KL|KLU}, CRFILE=filename,$
```

where:

segname
: Is identical to the SEGNAME in the Master File that contains the full description of the RDBMS table's columns (the remote Master File).

parent
: Is the parent of the segment.

`KL`
: Describes one-to-many relationships.

`KLU`
: Describes unique relationships.

filename
: Is the name of the remote Master File that contains the full description of the segment's fields.

Note:

- You may not use the attributes CRKEY and CRSEGNAME.
- If a Master File that contains a CRFILE cross-reference to a segment in another Master File does not contain a declaration for that segment in its own Access File, the Interface issues a FOC1351 message as a warning. The Interface then attempts to use the corresponding segment reference from the cross-referenced Access File. If the information in that Access File (such as the KEYFLD/IXFLD pair) can function correctly with the local Master File, the Interface continues processing. If not, it displays additional error messages, and processing terminates. The FOC1351 message should be considered only a warning unless accompanied by additional messages.
- You may not use Master Files containing remotely-described segments in CREATE FILE commands or MODIFY procedures.

SEGTYPEs KL and KLU describe segments whose field attributes are described in other Master Files that FOCUS may read at run-time. You can use remote segment descriptions for situations in which several Master Files introduce different views on the same collection of RDBMS tables. You describe the fields of one or several tables in one Master File, and refer to this first file from other Master Files without including all the field descriptions again.

The separately described segment must have the same name in the file in which it is defined and in the file that references it. The CRFILE attribute identifies the Master File that contains the complete segment definition. For example:

```
SEGNAME=DEPT, PARENT=EMP, SEGTYPE=KL, CRFILE=MAINFILE, $
```

The Interface obtains the descriptions of fields in the DEPT segment in this file from the DEPT segment in the MAINFILE Master File, where they must physically reside; DEPT cannot be a KL or KLU segment in MAINFILE. Similarly, a dynamic JOIN may not specify a KL or KLU segment as the cross-referenced segment in the target file.

Once you specify the CRFILE attribute in a Master File, that specification becomes the default for subsequently described segments; if you later wish to describe a segment locally (using the traditional method), you must re-specify the local filename using the CRFILE attribute, even though this is not technically a cross-reference. Obviously, the same holds true if you wish to change cross-referenced files from one segment to the next.

FOCUS reads only the field attributes from the segment, not the segment attributes. The MAINFILE Master File does not have to be the description of a real file. It does not need an Access File; it just needs the description of the named segment. Thus, you can set up one large Master File that contains field descriptions of all RDBMS tables or views you may use in reporting, even if you only use the large description for reference, not for reporting.

If you describe the root segment remotely, specify its SEGTYPE as KL.

If two or more segments in a FOCUS Master File represent the same RDBMS table, only one of these segments can have a remote segment description. This requirement is necessary in order to preserve unique SEGNAMEs within the local Master File (because a remote SEGNAME must be identical to its corresponding local SEGNAME).

Remote segment descriptions exist only for convenience. They save typing effort but offer no logical implications regarding parent-child relationships and their implementation.

A Master File that contains remote segment declarations must have its own Access File for defining the relationships between all of its segments (including the remote segments). This local Access File overrides the Access File corresponding to the CRFILE Master File, if one exists.

The following example shows the ECOURSE Master File with the COURSE segment described remotely:

```
FILENAME=ECOURSE     ,SUFFIX=SQLDS, $

SEGNAME=EMPINFO      ,SEGTYPE=S0, $
 FIELD=EMP_ID        ,ALIAS=EID         ,USAGE=A9    ,ACTUAL=A9,$
 FIELD=LAST_NAME     ,ALIAS=LN          ,USAGE=A15   ,ACTUAL=A15,$
 FIELD=FIRST_NAME    ,ALIAS=FN          ,USAGE=A10   ,ACTUAL=A10,$
 FIELD=HIRE_DATE     ,ALIAS=HDT         ,USAGE=YMD   ,ACTUAL=DATE,$
 FIELD=DEPARTMENT    ,ALIAS=DPT         ,USAGE=A10   ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY,ALIAS=CSAL        ,USAGE=P9.2  ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE  ,ALIAS=CJC         ,USAGE=A3    ,ACTUAL=A3,$
 FIELD=ED_HRS        ,ALIAS=OJT         ,USAGE=F6.2  ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN    ,ALIAS=BONUS_PLAN  ,USAGE=I4    ,ACTUAL=I4,$

SEGNAME=COURSE ,SEGTYPE=KL  ,PARENT=EMPINFO, CRFILE=COURSE,$
```

The corresponding Access File is:

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1,
   WRITE = YES,  DBSPACE = PUBLIC.SPACE0,$
SEGNAME = COURSE, TABLENAME = "USER1"."COURSE", KEYS = 2,
   WRITE = YES,     DBSPACE = PUBLIC.SPACE0,
     KEYFLD = EMP_ID, IXFLD = WHO,$
```

B Native SQL Commands

Topic:

- Native SQL

This appendix explains how to issue native SQL commands from within your FOCUS session.

Native SQL

Using the Interface, you can issue most SQL commands from within your FOCUS session. Unless you invoke Direct SQL Passthru (see Chapter 9, *Direct SQL Passthru*), you are limited to those commands that result in return codes rather than data. (For instance, DROP results in a return code, while SELECT returns data.)

The Interface passes the SQL command directly to the RDBMS for processing; it then detects the success or failure of the operation. If the operation fails, the Interface writes an appropriate error message to the terminal.

You can issue the SQL command from the FOCUS command level, include it in a FOCEXEC, or issue it using the Direct SQL Passthru facility.

Note:

- The restriction on commands that return data does not apply to commands issued through the Direct SQL Passthru facility.
- If the Interface was installed with IM=0, native SQL commands, except SQL COMMIT WORK and SQL ROLLBACK WORK, will be disabled. If this is the case at your site, error message FOC1638 will display when you attempt to execute a native SQL command. SQL SELECT commands are never disabled.

As with Interface commands, there are two forms of acceptable syntax for native SQL commands. Use the first form of syntax to invoke the Direct SQL Passthru facility. The first form is

```
SQL [target_db]
command [;]
[TABLE FILE SQLOUT]
[options]
END
```

where:

`target_db`
: Indicates the target RDBMS. Valid values are DB2 or SQLDS. Omit if you previously issued the SET SQLENGINE command.

`command`
: Is any SQL command.

`;`
: For SELECT requests only, a semicolon is required if you intend to specify additional FOCUS report options.

Direct SQL Passthru syntax does not include TSO, MVS, and CMS environmental qualifiers. The target_db is optional if you previously issued the SET SQLENGINE command. For a complete discussion of the Direct SQL Passthru facility and acceptable report options, see Chapter 9, *Direct SQL Passthru*.

Use the second form of syntax when not invoking the Direct SQL Passthru facility. This alternative syntax includes TSO, MVS, or CMS environmental qualifiers and the required SQL keyword

```
environment SQL sqlcommand
```

where:

`environment`
: Is one of the following: TSO, CMS, or MVS.

The qualifiers TSO and MVS are synonyms; you can use them interchangeably in any MVS environment.

Some SQL commands are:

`ACQUIRE DBSPACE`	Obtains a DBSPACE for the table you are creating.
`ALTER DBSPACE`	Changes DBSPACE characteristics.
`ALTER TABLE`	Adds one new column at the end of a table.
`COMMENT ON`	Places a comment in the REMARKS column of the SYSCATALOG or SYSCOLUMNS tables.
`COMMIT WORK`	Makes the current logical unit of work permanent.
`CONNECT`	Issues a Distributed Relational Database Architecture (DRDA) CONNECT.
`CREATE INDEX`	Creates an index on one or more columns of a table and assigns a name to the index.
`CREATE SYNONYM`	Defines an alternate name for a table or view.
`CREATE TABLE`	Creates a new table.
`CREATE VIEW`	Creates a view of one or more tables.
`DELETE`	Deletes one or more rows from a table.
`DROP DBSPACE`	Deletes a DBSPACE and its contents.
`DROP INDEX`	Deletes an index.
`DROP PROGRAM`	Deletes the application package associated with the named program.
`DROP SYNONYM`	Deletes the named synonym or alternate name of a table.
`DROP TABLE`	Deletes a table.
`DROP TABLESPACE`	Deletes a TABLESPACE and its contents.
`DROP VIEW`	Deletes a view.
`EXPLAIN`	Retrieves information about the structure and execution performance of SQL commands.
`GRANT`	Distributes specific privileges or authorities such as CREATE, SELECT, UPDATE.
`INSERT INTO`	Adds data to a table.
`LABEL`	Defines a label for a table or a column. You can use this label in query results.
`LOCK DBSPACE`	Locks a DBSPACE and its contents.

`LOCK TABLE`	Prevents concurrent changes to a table until a logical unit of work has completed (For instance, a TABLE request has completed or a COMMIT is encountered in a MODIFY procedure.).
`REVOKE`	Cancels a previously granted privilege.
`ROLLBACK WORK`	Backs out all changes made by the current logical unit of work.
`SELECT`	Retrieves data for the entire table or for specified columns. May not be used with the second form of syntax. See Chapter 9, *Direct SQL Passthru*, for the Direct SQL Passthru facility.
`UPDATE`	Changes data in a table.

Note: For more information about SQL commands, refer to the SQL Reference Manual for the appropriate RDBMS.

As an example, to drop the DB2 table, EMPLOYEE, issue the following from the FOCUS command level:

```
TSO SQL DROP TABLE EMPLOYEE
```

C File Descriptions and Tables

Topics:

This appendix consists of sample tables and views cited in previous chapters. It provides single-table Master and Access Files and CHECK FILE diagrams for:

- ADDRESS
- COURSE
- DEDUCT
- EMPINFO
- FUNDTRAN
- PAYINFO
- SALINFO

In addition, it includes the following multi-table Master and Access Files and CHECK FILE diagrams:

- ECOURSE
- EMPADD
- EMPFUND
- EMPPAY
- SALDUCT

An OCCURS segment is described in the following Master File and Access File with a CHECK FILE diagram:

- SALARY

This appendix also illustrates sample tables used in examples for the Direct SQL Passthru facility:

- DPBRANCH
- DPINVENT
- DPVENDOR

As discussed in Chapter 9, *Direct SQL Passthru*, the in-memory SQLOUT Master Files and the FOCUS views created with SQL PREPARE vary depending on the specified SELECT statements. They are not accessible for the above tables and are not provided in this appendix.

ADDRESS Sample

The ADDRESS table contains employees' home and bank addresses.

ADDRESS MASTER

```
FILENAME=ADDRESS  ,SUFFIX=SQLDS

SEGNAME=ADDRESS  ,SEGTYPE=S0,$
  FIELDNAME=ADDEID       ,ALIAS=EID     ,USAGE=A9    , ACTUAL = A9,$
  FIELDNAME=TYPE         ,ALIAS=AT      ,USAGE=A4    , ACTUAL = A4,$
  FIELDNAME=ADDRESS_LN1  ,ALIAS=LN1     ,USAGE=A20   , ACTUAL = A20,$
  FIELDNAME=ADDRESS_LN2  ,ALIAS=LN2     ,USAGE=A20   , ACTUAL = A20,$
  FIELDNAME=ADDRESS_LN3  ,ALIAS=LN3     ,USAGE=A20   , ACTUAL = A20,$
  FIELDNAME=ACCTNUMBER   ,ALIAS=ANO     ,USAGE=I9L   , ACTUAL = I4,$
```

ADDRESS FOCSQL

```
SEGNAME = ADDRESS, TABLENAME = "USER1"."ADDRESS", KEYS = 2, WRITE = YES,
DBSPACE = PUBLIC.SPACE0,$
```

ADDRESS Diagram

```
check file address pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   1  ( REAL=    1  VIRTUAL=   0 )
  NUMBER OF FIELDS=     6  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   77
 SECTION 01
              STRUCTURE OF SQLDS    FILE ADDRESS  ON 06/17/93 AT 13.51.17

          ADDRESS
  01      S0
 **************
 *ADDEID      **
 *TYPE        **
 *ADDRESS_LN1 **
 *ADDRESS_LN2 **
 *            **
 ***************
  **************
```

COURSE Sample

The COURSE table lists the courses that the employees attended.

COURSE MASTER

```
FILENAME=COURSE ,SUFFIX=SQLDS,$

SEGNAME=COURSE  ,SEGTYPE=S0  ,$
 FIELD=CNAME   ,ALIAS=COURSE_NAME ,USAGE=A15, ACTUAL=A15,$
 FIELD=WHO     ,ALIAS=EMP_NO      ,USAGE=A9,  ACTUAL=A9,$
 FIELD=GRADE   ,ALIAS=GRADE       ,USAGE=A1,  ACTUAL=A1, MISSING=ON,$
 FIELD=YR_TAKEN,ALIAS=YR_TAKEN    ,USAGE=A2,  ACTUAL=A2,$
 FIELD=QTR     ,ALIAS=QUARTER     ,USAGE=A1,  ACTUAL=A1,$
```

COURSE FOCSQL

```
SEGNAME = COURSE, TABLENAME = "USER1"."COURSE", KEYS = 2, WRITE = YES,
    DBSPACE = PUBLIC.SPACE0, $
```

COURSE Diagram

```
check file course pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   1  ( REAL=    1  VIRTUAL=   0 )
  NUMBER OF FIELDS=     5  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   28
 SECTION 01
              STRUCTURE OF SQLDS    FILE COURSE   ON 06/17/93 AT 13.51.41

          COURSE
  01      S0
 **************
 *CNAME       **
 *WHO         **
 *GRADE       **
 *YR_TAKEN    **
 *            **
 ***************
  **************
```

DEDUCT Sample

The DEDUCT table contains data on monthly pay deductions.

DEDUCT MASTER

```
FILENAME=DEDUCT ,  SUFFIX=SQLDS,$

SEGNAME=DEDUCT   ,SEGTYPE=S0,$
  FIELDNAME=DEDEID  ,ALIAS=EID ,USAGE=A9       ,ACTUAL = A9,$
  FIELDNAME=DEDDATE ,ALIAS=PD  ,USAGE=YMD      ,ACTUAL = DATE,$
  FIELDNAME=DED_CODE,ALIAS=DC  ,USAGE=A4       ,ACTUAL = A4,$
  FIELDNAME=DED_AMT ,ALIAS=DA  ,USAGE=D12.2M   ,ACTUAL = D8,$
```

DEDUCT FOCSQL

```
SEGNAME = DEDUCT, TABLENAME = "USER1"."DEDUCT",  KEYS = 3, WRITE = YES,
    KEYORDER = HIGH, DBSPACE = PUBLIC.SPACE0,$
```

DEDUCT Diagram

```
check file deduct pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   1  ( REAL=    1  VIRTUAL=   0 )
  NUMBER OF FIELDS=     4  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   25
 SECTION 01
              STRUCTURE OF SQLDS    FILE DEDUCT   ON 06/16/93 AT 09.42.45

          DEDUCT
  01      S0
 **************
 *DEDEID      **
 *DEDDATE     **
 *DED_CODE    **
 *DED_AMT     **
 *            **
 ***************
  **************
```

EMPINFO Sample

The EMPINFO table contains employee IDs, names, positions, and current salary information.

EMPINFO MASTER

```
FILENAME=EMPINFO      ,SUFFIX=SQLDS,$

SEGNAME=EMPINFO       ,SEGTYPE=S0,$
 FIELD=EMP_ID         ,ALIAS=EID         ,USAGE=A9    ,ACTUAL=A9,$
 FIELD=LAST_NAME      ,ALIAS=LN          ,USAGE=A15   ,ACTUAL=A15,$
 FIELD=FIRST_NAME     ,ALIAS=FN          ,USAGE=A10   ,ACTUAL=A10,$
 FIELD=HIRE_DATE      ,ALIAS=HDT         ,USAGE=YMD   ,ACTUAL=DATE,$
 FIELD=DEPARTMENT     ,ALIAS=DPT         ,USAGE=A10   ,ACTUAL=A10,
    MISSING=ON,$
 FIELD=CURRENT_SALARY ,ALIAS=CSAL        ,USAGE=P9.2  ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE   ,ALIAS=CJC         ,USAGE=A3    ,ACTUAL=A3,$
 FIELD=ED_HRS         ,ALIAS=OJT         ,USAGE=F6.2  ,ACTUAL=F4,   MISSING=ON,$
 FIELD=BONUS_PLAN     ,ALIAS=BONUS_PLAN  ,USAGE=I4    ,ACTUAL=I4,$
```

EMPINFO FOCSQL

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1,
 WRITE = YES,  DBSPACE = PUBLIC.SPACE0,$
```

EMPINFO Diagram

```
check file empinfo pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   1  ( REAL=    1  VIRTUAL=   0 )
  NUMBER OF FIELDS=     9  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   63
 SECTION 01
              STRUCTURE OF SQLDS    FILE EMPINFO  ON 06/16/93 AT 09.44.03

          EMPINFO
  01      S0
 **************
 *EMP_ID      **
 *LAST_NAME   **
 *FIRST_NAME  **
 *HIRE_DATE   **
 *            **
 ***************
  **************
```

FUNDTRAN Sample

The FUNDTRAN table contains data about the employees' direct deposit accounts.

FUNDTRAN MASTER

```
FILENAME=FUNDTRAN ,SUFFIX=SQLDS

SEGNAME=FUNDTRAN, SEGTYPE=S0,$
 FIELDNAME=WHO          ,ALIAS=EID   ,USAGE=A9    ,ACTUAL=A9,$
 FIELDNAME=BANK_NAME    ,ALIAS=BN    ,USAGE=A20   ,ACTUAL=A20,$
 FIELDNAME=BANK_CODE    ,ALIAS=BC    ,USAGE=I6S   ,ACTUAL=I4,$
 FIELDNAME=BANK_ACCT    ,ALIAS=BA    ,USAGE=I9S   ,ACTUAL=I4,$
 FIELDNAME=EFFECT_DATE  ,ALIAS=EDATE ,USAGE=YMD   ,ACTUAL=DATE,$
```

FUNDTRAN FOCSQL

```
SEGNAME = FUNDTRAN, TABLENAME = "USER1"."FUNDTRAN", KEYS = 1,
    WRITE = YES, DBSPACE = PUBLIC.SPACE0,$
```

FUNDTRAN Diagram

```
check file fundtran pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   1  ( REAL=    1  VIRTUAL=   0 )
  NUMBER OF FIELDS=     5  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   41
 SECTION 01
              STRUCTURE OF SQLDS    FILE FUNDTRAN ON 06/17/93 AT 13.53.15

          FUNDTRAN
  01      S0
 **************
 *WHO         **
 *BANK_NAME   **
 *BANK_CODE   **
 *BANK_ACCT   **
 *            **
 ***************
  **************
```

PAYINFO Sample

The PAYINFO table contains the employees' salary history.

PAYINFO MASTER

```
FILENAME=PAYINFO,  SUFFIX=SQLDS,$

SEGNAME=PAYINFO  ,SEGTYPE=S0,$
  FIELDNAME=PAYEID  ,ALIAS=EID ,USAGE=A9        ,ACTUAL=A9,$
  FIELDNAME=DAT_INC ,ALIAS=DI  ,USAGE=YMD       ,ACTUAL=DATE,$
  FIELDNAME=PCT_INC ,ALIAS=PI  ,USAGE=F6.2      ,ACTUAL=F4,$
  FIELDNAME=SALARY  ,ALIAS=SAL ,USAGE=D12.2M    ,ACTUAL=D8,$
  FIELDNAME=JOBCODE ,ALIAS=JBC ,USAGE=A3        ,ACTUAL=A3,$
```

PAYINFO FOCSQL

```
SEGNAME = PAYINFO, TABLENAME = "USER1"."PAYINFO", KEYS = 2, WRITE = YES,
    DBSPACE = PUBLIC.SPACE0,$
```

PAYINFO Diagram

```
check file payinfo pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   1  ( REAL=     1  VIRTUAL=   0 )
  NUMBER OF FIELDS=     5  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   28
 SECTION 01
              STRUCTURE OF SQLDS    FILE PAYINFO  ON 06/16/93 AT 09.44.31

          PAYINFO
  01      S0
 **************
 *PAYEID      **
 *DAT_INC     **
 *PCT_INC     **
 *SALARY      **
 *            **
 ***************
  **************
```

SALINFO Sample

The SALINFO table contains data on the employees' monthly pay.

SALINFO MASTER

```
FILENAME=SALINFO,  SUFFIX=SQLDS,$

SEGNAME=SALINFO  ,SEGTYPE=S0,$
 FIELDNAME=SALEID   ,ALIAS=EID     ,USAGE=A9      ,ACTUAL=A9,$
 FIELDNAME=PAY_DATE ,ALIAS=PD      ,USAGE=YMD     ,ACTUAL=DATE,$
 FIELDNAME=GROSS    ,ALIAS=MO_PAY  ,USAGE=D12.2M  ,ACTUAL=D8,$
```

SALINFO FOCSQL

```
SEGNAME = SALINFO, TABLENAME = "USER1"."SALINFO", KEYS = 2,
 WRITE = YES,  KEYORDER = HIGH, DBSPACE = PUBLIC.SPACE0,$
```

SALINFO Diagram

```
check file salinfo pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   1  ( REAL=    1  VIRTUAL=   0 )
  NUMBER OF FIELDS=     3  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   21
 SECTION 01
              STRUCTURE OF SQLDS    FILE SALINFO  ON 06/17/93 AT 13.52.04

          SALINFO
  01      S0
 **************
 *SALEID      **
 *PAY_DATE    **
 *GROSS       **
 *            **
 *            **
 ***************
  **************
```

ECOURSE Sample

The ECOURSE view accesses the EMPINFO and COURSE tables.

ECOURSE MASTER

```
FILENAME=ECOURSE     ,SUFFIX=SQLDS, $

SEGNAME=EMPINFO      ,SEGTYPE=S0, $
 FIELD=EMP_ID        ,ALIAS=EID         ,USAGE=A9    ,ACTUAL=A9,$
 FIELD=LAST_NAME     ,ALIAS=LN          ,USAGE=A15   ,ACTUAL=A15,$
 FIELD=FIRST_NAME    ,ALIAS=FN          ,USAGE=A10   ,ACTUAL=A10,$
 FIELD=HIRE_DATE     ,ALIAS=HDT         ,USAGE=YMD   ,ACTUAL=DATE,$
 FIELD=DEPARTMENT    ,ALIAS=DPT         ,USAGE=A10   ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY,ALIAS=CSAL        ,USAGE=P9.2  ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE  ,ALIAS=CJC         ,USAGE=A3    ,ACTUAL=A3,$
 FIELD=ED_HRS        ,ALIAS=OJT         ,USAGE=F6.2  ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN    ,ALIAS=BONUS_PLAN  ,USAGE=I4    ,ACTUAL=I4,$

SEGNAME=COURSE ,SEGTYPE=S0  ,PARENT=EMPINFO,$
 FIELD=CNAME   ,ALIAS=COURSE_NAME ,USAGE=A15, ACTUAL=A15,$
 FIELD=WHO     ,ALIAS=EMP_NO      ,USAGE=A9,  ACTUAL=A9,$
 FIELD=GRADE   ,ALIAS=GRADE       ,USAGE=A1,  ACTUAL=A1, MISSING=ON,$
 FIELD=YR_TAKEN,ALIAS=YR_TAKEN    ,USAGE=A2,  ACTUAL=A2,$
 FIELD=QTR     ,ALIAS=QUARTER     ,USAGE=A1,  ACTUAL=A1,$
```

ECOURSE FOCSQL

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1,
   WRITE = YES,  DBSPACE = PUBLIC.SPACE0,$
SEGNAME = COURSE, TABLENAME = "USER1"."COURSE", KEYS = 2,
   WRITE = YES,     DBSPACE = PUBLIC.SPACE0,
     KEYFLD = EMP_ID, IXFLD = WHO,$
```

ECOURSE Diagram

```
check file ecourse pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   2  ( REAL=    2  VIRTUAL=   0 )
  NUMBER OF FIELDS=    14  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   91
 SECTION 01
              STRUCTURE OF SQLDS    FILE ECOURSE  ON 06/16/93 AT 09.43.01

          EMPINFO
  01      S0
 **************
 *EMP_ID      **
 *LAST_NAME   **
 *FIRST_NAME  **
 *HIRE_DATE   **
 *            **
 ***************
  **************
        I
        I
        I
        I COURSE
  02    I S0
 **************
 *CNAME       **
 *WHO         **
 *GRADE       **
 *YR_TAKEN    **
 *            **
 ***************
  **************
```

EMPADD Sample

The EMPADD view accesses the EMPINFO and ADDRESS tables.

EMPADD MASTER

```
FILENAME=EMPADD,   SUFFIX=SQLDS,$

SEGNAME=EMPINFO  ,SEGTYPE=S0,$
 FIELD=EMP_ID        ,ALIAS=EID           ,USAGE=A9       ,ACTUAL=A9,$
 FIELD=LAST_NAME     ,ALIAS=LN            ,USAGE=A15      ,ACTUAL=A15,$
 FIELD=FIRST_NAME    ,ALIAS=FN            ,USAGE=A10      ,ACTUAL=A10,$
 FIELD=HIRE_DATE     ,ALIAS=HDT           ,USAGE=YMD      ,ACTUAL=DATE,$
 FIELD=DEPARTMENT    ,ALIAS=DPT           ,USAGE=A10      ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY,ALIAS=CSAL          ,USAGE=P9.2     ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE  ,ALIAS=CJC           ,USAGE=A3       ,ACTUAL=A3,$
 FIELD=ED_HRS        ,ALIAS=OJT           ,USAGE=F6.2     ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN    ,ALIAS=BONUS_PLAN    ,USAGE=I4       ,ACTUAL=I4,$

SEGNAME=ADDRESS  ,SEGTYPE=S0, PARENT = EMPINFO,$
  FIELD=ADDEID           ,ALIAS=EID  ,USAGE=A9      ,ACTUAL=A9,$
  FIELD=TYPE             ,ALIAS=AT   ,USAGE=A4      ,ACTUAL=A4,$
  FIELD=ADDRESS_LN1      ,ALIAS=LN1  ,USAGE=A20     ,ACTUAL=A20,$
  FIELD=ADDRESS_LN2      ,ALIAS=LN2  ,USAGE=A20     ,ACTUAL=A20,$
  FIELD=ADDRESS_LN3      ,ALIAS=LN3  ,USAGE=A20     ,ACTUAL=A20,$
  FIELD=ACCTNUMBER       ,ALIAS=ANO  ,USAGE=I9L     ,ACTUAL=I4,$
```

EMPADD FOCSQL

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1, WRITE = YES,
     DBSPACE = PUBLIC.SPACE0,$
SEGNAME = ADDRESS, TABLENAME = "USER1"."ADDRESS", KEYS = 2, WRITE = YES,
     DBSPACE = PUBLIC.SPACE0,
     KEYFLD = EMP_ID, IXFLD = ADDEID,$
```

EMPADD Diagram

```
check file empadd pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   2  ( REAL=    2  VIRTUAL=   0 )
  NUMBER OF FIELDS=    15  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=  140
 SECTION 01
              STRUCTURE OF SQLDS    FILE EMPADD   ON 06/16/93 AT 09.43.16

          EMPINFO
  01      S0
 **************
 *EMP_ID      **
 *LAST_NAME   **
 *FIRST_NAME  **
 *HIRE_DATE   **
 *            **
 ***************
  **************
        I
        I
        I
        I ADDRESS
  02    I S0
 **************
 *ADDEID      **
 *TYPE        **
 *ADDRESS_LN1 **
 *ADDRESS_LN2 **
 *            **
 ***************
  **************
```

EMPFUND Sample

The EMPFUND view accesses the EMPINFO and FUNDTRAN tables.

EMPFUND MASTER

```
FILENAME=EMPFUND ,SUFFIX=SQLDS,$

SEGNAME=EMPINFO  ,SEGTYPE=S0,$
 FIELD=EMP_ID        ,ALIAS=EID           ,USAGE=A9       ,ACTUAL=A9,$
 FIELD=LAST_NAME     ,ALIAS=LN            ,USAGE=A15      ,ACTUAL=A15,$
 FIELD=FIRST_NAME    ,ALIAS=FN            ,USAGE=A10      ,ACTUAL=A10,$
 FIELD=HIRE_DATE     ,ALIAS=HDT           ,USAGE=YMD      ,ACTUAL=DATE,$
 FIELD=DEPARTMENT    ,ALIAS=DPT           ,USAGE=A10      ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY,ALIAS=CSAL          ,USAGE=P9.2     ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE  ,ALIAS=CJC           ,USAGE=A3       ,ACTUAL=A3,$
 FIELD=ED_HRS        ,ALIAS=OJT           ,USAGE=F6.2     ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN    ,ALIAS=BONUS_PLAN    ,USAGE=I4       ,ACTUAL=I4,$

SEGNAME=FUNDTRAN, SEGTYPE=U, PARENT=EMPINFO,$
 FIELDNAME=WHO          ,ALIAS=EID   ,USAGE=A9    ,ACTUAL=A9,$
 FIELDNAME=BANK_NAME    ,ALIAS=BN    ,USAGE=A20   ,ACTUAL=A20,$
 FIELDNAME=BANK_CODE    ,ALIAS=BC    ,USAGE=I6S   ,ACTUAL=I4,$
 FIELDNAME=BANK_ACCT    ,ALIAS=BA    ,USAGE=I9S   ,ACTUAL=I4,$
 FIELDNAME=EFFECT_DATE  ,ALIAS=EDATE ,USAGE=YMD   ,ACTUAL=DATE,$
```

EMPFUND FOCSQL

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1, WRITE = YES,
     DBSPACE = PUBLIC.SPACE0,$
SEGNAME = FUNDTRAN, TABLENAME = "USER1"."FUNDTRAN", KEYS = 1,
    WRITE = YES, DBSPACE = PUBLIC.SPACE0,
    KEYFLD = EMP_ID, IXFLD = WHO,$
```

EMPFUND Diagram

```
check file empfund pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   2  ( REAL=    2  VIRTUAL=   0 )
  NUMBER OF FIELDS=    14  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=  104
 SECTION 01
              STRUCTURE OF SQLDS    FILE EMPFUND  ON 06/16/93 AT 09.43.45

          EMPINFO
  01      S0
 **************
 *EMP_ID      **
 *LAST_NAME   **
 *FIRST_NAME  **
 *HIRE_DATE   **
 *            **
 ***************
  **************
        I
        I
        I
        I FUNDTRAN
  02    I U
 **************
 *WHO         *
 *BANK_NAME   *
 *BANK_CODE   *
 *BANK_ACCT   *
 *            *
 **************
```

EMPPAY Sample

The EMPAY view accesses the EMPINFO and PAYINFO tables.

EMPPAY MASTER

```
FILENAME=EMPPAY  ,SUFFIX=SQLDS,$

SEGNAME=EMPINFO  ,SEGTYPE=S0,$
 FIELD=EMP_ID        ,ALIAS=EID           ,USAGE=A9       ,ACTUAL=A9,$
 FIELD=LAST_NAME     ,ALIAS=LN            ,USAGE=A15      ,ACTUAL=A15,$
 FIELD=FIRST_NAME    ,ALIAS=FN            ,USAGE=A10      ,ACTUAL=A10,$
 FIELD=HIRE_DATE     ,ALIAS=HDT           ,USAGE=YMD      ,ACTUAL=DATE,$
 FIELD=DEPARTMENT    ,ALIAS=DPT           ,USAGE=A10      ,ACTUAL=A10,
   MISSING=ON,$
 FIELD=CURRENT_SALARY,ALIAS=CSAL          ,USAGE=P9.2     ,ACTUAL=P4,$
 FIELD=CURR_JOBCODE  ,ALIAS=CJC           ,USAGE=A3       ,ACTUAL=A3,$
 FIELD=ED_HRS        ,ALIAS=OJT           ,USAGE=F6.2     ,ACTUAL=F4,
   MISSING=ON,$
 FIELD=BONUS_PLAN    ,ALIAS=BONUS_PLAN    ,USAGE=I4       ,ACTUAL=I4,$
SEGNAME=PAYINFO  ,SEGTYPE=S0, PARENT=EMPINFO, $
  FIELDNAME=PAYEID  ,ALIAS=EID ,USAGE=A9        ,ACTUAL=A9,$
  FIELDNAME=DAT_INC ,ALIAS=DI  ,USAGE=YMD       ,ACTUAL=DATE,$
  FIELDNAME=PCT_INC ,ALIAS=PI  ,USAGE=F6.2      ,ACTUAL=F4,$
  FIELDNAME=SALARY  ,ALIAS=SAL ,USAGE=D12.2M    ,ACTUAL=D8,$
  FIELDNAME=JOBCODE ,ALIAS=JBC ,USAGE=A3        ,ACTUAL=A3,$
```

EMPPAY FOCSQL

```
SEGNAME = EMPINFO, TABLENAME = "USER1"."EMPINFO", KEYS = 1, WRITE = YES,
     DBSPACE = PUBLIC.SPACE0,$
SEGNAME = PAYINFO, TABLENAME = "USER1"."PAYINFO", KEYS = 2, WRITE = YES,
     DBSPACE = PUBLIC.SPACE0,
     KEYFLD = EMP_ID, IXFLD = PAYEID, $
```

EMPPAY Diagram

```
check file emppay pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   2  ( REAL=    2  VIRTUAL=   0 )
  NUMBER OF FIELDS=    14  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   91
 SECTION 01
              STRUCTURE OF SQLDS    FILE EMPPAY   ON 06/16/93 AT 09.44.14

          EMPINFO
  01      S0
 **************
 *EMP_ID      **
 *LAST_NAME   **
 *FIRST_NAME  **
 *HIRE_DATE   **
 *            **
 ***************
  **************
        I
        I
        I
        I PAYINFO
  02    I S0
 **************
 *PAYEID      **
 *DAT_INC     **
 *PCT_INC     **
 *SALARY      **
 *            **
 ***************
  **************
```

SALDUCT Sample

The SALDUCT view accesses the SALINFO and DEDUCT tables.

SALDUCT MASTER

```
FILENAME=SALDUCT,  SUFFIX=SQLDS,$

SEGNAME=SALINFO  ,SEGTYPE=S0,$
 FIELDNAME=SALEID   ,ALIAS=EID     ,USAGE=A9      ,ACTUAL=A9,$
 FIELDNAME=PAY_DATE ,ALIAS=PD      ,USAGE=YMD     ,ACTUAL=DATE,$
 FIELDNAME=GROSS    ,ALIAS=MO_PAY  ,USAGE=D12.2M  ,ACTUAL=D8,$

SEGNAME=DEDUCT   ,SEGTYPE=S0, PARENT =SALINFO,$
  FIELDNAME=DEDEID  ,ALIAS=EID ,USAGE=A9       ,ACTUAL = A9,$
  FIELDNAME=DEDDATE ,ALIAS=PD  ,USAGE=YMD      ,ACTUAL = DATE,$
  FIELDNAME=DED_CODE,ALIAS=DC  ,USAGE=A4       ,ACTUAL = A4,$
  FIELDNAME=DED_AMT ,ALIAS=DA  ,USAGE=P9.2     ,ACTUAL = P4,$
```

SALDUCT FOCSQL

```
SEGNAME = SALINFO, TABLENAME = "USER1"."SALINFO", KEYS = 2, WRITE = YES,
    KEYORDER = HIGH, DBSPACE = PUBLIC.SPACE0,$
SEGNAME = DEDUCT, TABLENAME = "USER1"."DEDUCT",  KEYS = 3, WRITE = YES,
    KEYORDER = HIGH, DBSPACE = PUBLIC.SPACE0,
    KEYFLD = SALEID/PAY_DATE , IXFLD = DEDEID/DEDDATE,$
```

SALDUCT Diagram

```
check file salduct pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   2  ( REAL=    2  VIRTUAL=   0 )
  NUMBER OF FIELDS=     7  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=   42
 SECTION 01
              STRUCTURE OF SQLDS    FILE SALDUCT  ON 06/17/93 AT 13.52.47

          SALINFO
  01      S0
 **************
 *SALEID      **
 *PAY_DATE    **
 *GROSS       **
 *            **
 *            **
 ***************
  **************
       I
       I
       I
       I DEDUCT
  02   I S0
 **************
 *DEDEID      **
 *DEDDATE     **
 *DED_CODE    **
 *DED_AMT     **
 *            **
 ***************
  **************
```

SALARY Sample

The SALARY table contains data on salary and monthly pay deductions.

SALARY MASTER

```
FILENAME=SALARY, SUFFIX=SQLDS,$

SEGNAME=SALARY,  SEGTYPE=S0,$
 FIELD=EMPID,    ALIAS=EMPID,    USAGE=A7,   ACTUAL=A7,$
 FIELD=EMPNAME,  ALIAS=EMPNAME,  USAGE=A10,  ACTUAL=A10,$
 FIELD=SALARY,   ALIAS=PAY,      USAGE=P9.2, ACTUAL=P4,$

 FIELD=DEDUCT1,  ALIAS=DEDUCT1,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT2,  ALIAS=DEDUCT2,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT3,  ALIAS=DEDUCT3,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT4,  ALIAS=DEDUCT4,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT5,  ALIAS=DEDUCT5,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT6,  ALIAS=DEDUCT6,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT7,  ALIAS=DEDUCT7,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT8,  ALIAS=DEDUCT8,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT9,  ALIAS=DEDUCT9,  USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT10, ALIAS=DEDUCT10, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT11, ALIAS=DEDUCT11, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=DEDUCT12, ALIAS=DEDUCT12, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$

SEGNAME=OCC, PARENT=SALARY, POSITION=DEDUCT1, OCCURS=12,$
 FIELD=TAX,      ALIAS=TAXDEDUC, USAGE=P9.2, ACTUAL=P8, MISSING=ON,$
 FIELD=ORDER,    ALIAS=ORDER,    USAGE=I4,   ACTUAL=I4,$
```

SALARY FOCSQL

```
SEGNAME = SALARY, TABLENAME = "USER1"."SALARY", KEYS = 1,
WRITE = NO,   DBSPACE = PUBLIC.SPACE0,$
```

SALARY Diagram With OCCURS Segment

```
check file salary pict
  NUMBER OF ERRORS=     0
  NUMBER OF SEGMENTS=   2  ( REAL=    2  VIRTUAL=   0 )
  NUMBER OF FIELDS=     6  INDEXES=   0  FILES=     1
  TOTAL LENGTH OF ALL FIELDS=  129
 SECTION 01
              STRUCTURE OF SQLDS    FILE SALARY   ON 06/16/93 AT 09.44.51

          SALARY
  01      S0
 **************
 *EMPID       **
 *EMPNAME     **
 *SALARY      **
 *DEDUCT      **
 *            **
 ***************
  **************
        I
        I
        I
        I OCC
  02    I S0
 **************
 *TAX         **
 *ORDER       **
 *            **
 *            **
 *            **
 ***************
  **************
```

DPBRANCH Sample

The DPBRANCH table contains branch information.

DPBRANCH Table Definition

```
CREATE TABLE DPBRANCH
  (BRANCH_NUMBER    SMALLINT     NOT NULL,
   BRANCH_NAME      CHAR(5)      NOT NULL,
   BRANCH_MANAGER   CHAR(5)      NOT NULL,
   BRANCH_CITY      CHAR(5)      NOT NULL)
IN PUBLIC.SPACE0 ;
```

DPBRANCH Contents

```
SELECT * FROM DPBRANCH;

BRANCH_NUMBER BRANCH_NAME BRANCH_MANAGER BRANCH_CITY
------------- ----------- -------------- -----------
            1 WEST        PIAF           NY
            2 EAST        SMITH          NY
            3 NORTH       AMES           NY
            4 EAST        ALVIN          MIAMI
            5 WEST        FIELD          MIAMI
```

DPINVENT Sample

The DPINVENT table contains inventory information.

DPINVENT Table Definition

```
CREATE TABLE DPINVENT
  (BRANCH_NUMBER    SMALLINT     NOT NULL,
   VENDOR_NUMBER    SMALLINT     NOT NULL,
   PRODUCT          CHAR(5)      NOT NULL,
   NUMBER_OF_UNITS  SMALLINT     NOT NULL,
   PER_UNIT_VALUE   DECIMAL(9,2) NOT NULL)
IN PUBLIC.SPACE0 ;
```

DPINVENT Contents

```
SELECT * FROM DPINVENT;

BRANCH_NUMBER VENDOR_NUMBER PRODUCT NUMBER_OF_UNITS PER_UNIT_VALUE
------------- ------------- ------- --------------- --------------
            1             1 RADIO                 5          12.95
            2             1 RADIO                 5          12.95
            3             1 RADIO                 5          12.95
            4             1 RADIO                 5          12.95
            1             3 MICRO                 6         110.00
            4             3 MICRO                 2         110.00
            2             3 MICRO                 1         110.00
            2             3 MICRO                 2         110.00
```

DPVENDOR Sample

The DPVENDOR table contains vendor information.

DPVENDOR Table Definition

```
CREATE TABLE DPVENDOR
  (VENDOR_NUMBER    SMALLINT     NOT NULL,
   VENDOR_NAME      CHAR(5)      NOT NULL,
   VENDOR_CITY      CHAR(5)      NOT NULL)
IN PUBLIC.SPACE0 ;
```

DPVENDOR Contents

```
SELECT * FROM DPVENDOR;

VENDOR_NUMBER VENDOR_NAME VENDOR_CITY
------------- ----------- -----------
            1 ACME        NY
            2 STAR        OMAHA
            3 MMT         NY
            4 PIKE        MIAMI
```

D Tracing Interface Processing

Topics:

- FSTRACE
- FSTRACE3
- FSTRACE4
- Disabling Traces
- Batch Trace Allocation

The Interface communicates with the RDBMS through SQL statements it generates on your behalf. You can view these SQL statements with the Interface FSTRACE facilities. Traces are helpful for debugging procedures or for Interface performance analysis. The Interface FSTRACE facilities are easy to invoke, require no changes to either the Interface or FOCUS request, and have no effect on how the Interface functions.

There are three types of FSTRACE. You invoke each with either an ALLOCATE or DYNAM statement in MVS, or with a FILEDEF statement in CMS. The following chart lists the associated DDNAME and function for each trace:

DDNAME	**Trace Function**
FSTRACE	Traces Interface SQL statements, RDBMS return codes, COMMIT and ROLLBACK commands, and SQL cursor operations such as OPEN, FETCH and CLOSE.
FSTRACE3	Displays Interface-to-RDBMS aggregation and join analysis.
FSTRACE4	Traces Interface SQL statements resulting from report requests, MODIFY operations, and CREATE FILE commands.

You can allocate all or any combination of these traces during your FOCUS session or in batch. You can display the results online or store them in a file or sequential data set.

Note: The trace facilities are intended for use in query optimization and problem debugging. Application programs should not be written to depend on the format or content of any trace, as they may change in later releases of the Interface.

FSTRACE

FSTRACE records SQL statements, RDBMS return codes, COMMIT and ROLLBACK commands, and SQL cursor operations such as PREPARE, OPEN, FETCH, and CLOSE. You can use FSTRACE with all FOCUS report requests, MODIFY requests, and with native SQL commands. You can store trace information in an MVS sequential data set or CMS file, or you can display it online at the terminal. To capture trace data, issue the appropriate command from the FOCUS command level

```
{MVS|TSO} ALLOC F(FSTRACE) DA('userid.FSTRACE') SHR REUSE LRECL(80) RECFM(F)
```

or

```
DYNAM ALLOC FILE FSTRACE DATASET userid.FSTRACE SHR REUSE LRECL 80 RECFM F
```

or:

```
CMS FILEDEF FSTRACE DISK FSTRACE DATA A (LRECL 80 RECFM F
```

Note: DCB attributes for LRECL and RECFM are required, as shown in the preceding examples.

To view the trace information, use the system editor or the FOCUS TED editor.

To display trace data at the terminal, issue the appropriate command from the FOCUS command level

```
{MVS|TSO} ALLOC F(FSTRACE) DA(*) SHR REU
```

or

```
DYNAM ALLOC FILE FSTRACE DA *
```

or:

```
CMS FILEDEF FSTRACE TERMINAL
```

FSTRACE3

FSTRACE3 indicates whether the Interface has successfully transferred aggregation and join operations to the RDBMS. You can use it only for FOCUS reporting operations such as TABLE, GRAPH, and MATCH FILE.

To capture trace data, issue the appropriate command from the FOCUS command level

```
{MVS|TSO} ALLOC F(FSTRACE3) DA('userid.FSTRACE3') MOD REUSE LRECL(80) RECFM(F)
```

or

```
DYNAM ALLOC FILE FSTRACE3 DATASET userid.FSTRACE3 -
   MOD REUSE LRECL 80 RECFM F
```

or:

```
CMS FILEDEF FSTRACE3 DISK FSTRACE3 DATA A (LRECL 80 RECFM F DISP MOD
```

Note: You must specify the MOD parameter in order to produce a complete trace listing. The FSTRACE3 data set is opened and closed for each trace record. Without a MOD disposition parameter requests that produce multiple statements store only the last statement in the data set.

To display trace data on the terminal, issue the appropriate command from the FOCUS command level

```
{MVS|TSO} ALLOC F(FSTRACE3) DA(*) SHR REU
```

or

```
DYNAM ALLOC FILE FSTRACE3 DA *
```

or:

```
CMS FILEDEF FSTRACE3 TERMINAL
```

Note: When the Interface is able to pass all join, sort, and aggregation operations to the RDBMS, it does not populate FSTRACE3. In this case, the message “AGGREGATION DONE ...” appears either at the terminal or in the job output, whichever is appropriate.

FSTRACE4

FSTRACE4 records SQL SELECT statements generated by the Interface for FOCUS report requests, MODIFY procedures, or Direct SQL Passthru SELECT requests. It also records the SQL Data Definition Language (DDL) statements generated by the CREATE FILE command. You can display the trace information online or store it in a file or sequential data set. This technique can save you time and effort when you create tables and indexes.

The Interface terminates its generated SQL SELECT statements with a semicolon. Therefore, you can submit them to the RDBMS for processing, interactively or in batch, without any modifications. Thus, you can use FSTRACE4 for debugging, performance tuning, and for capturing SQL Data Definition and Data Manipulation statements to reuse.

To capture trace data, issue the appropriate command from the FOCUS command level

```
{MVS|TSO} ALLOC F(FSTRACE4) DA('userid.FSTRACE4') MOD REUSE LRECL(80) RECFM(F)
```

or

```
DYNAM ALLOC FILE FSTRACE4 DATASET userid.FSTRACE4 -
   MOD REUSE LRECL 80 RECFM F
```

or:

```
CMS FILEDEF FSTRACE4 DISK FSTRACE4 DATA A (LRECL 80 RECFM F DISP MOD
```

Note: You must specify the MOD parameter in order to produce a complete trace listing. The FSTRACE4 data set is opened and closed for each trace record. Without a MOD disposition parameter, requests that produce multiple SELECT statements store only the last statement in the data set.

To display trace data on the terminal, issue the appropriate command from the FOCUS command level

```
{MVS|TSO} ALLOC F(FSTRACE4) DA(*) SHR REU
```

or

```
DYNAM ALLOC FILE FSTRACE4 DA *
```

or:

```
CMS FILEDEF FSTRACE4 TERMINAL
```

Disabling Traces

To disable an Interface trace, clear the associated allocation

```
{MVS|TSO} FREE F(ddname)
```

or

```
DYNAM FREE FILE ddname
```

or

```
CMS FILEDEF ddname CLEAR
```

where:

ddname
Is FSTRACE, FSTRACE3, or FSTRACE4.

Batch Trace Allocation

You can write FSTRACE, FSTRACE3, and FSTRACE4 results to SYSOUT. BLKSIZE information is optional, but should be compatible with other FSTRACE formats:

```
//FSTRACE   DD   SYSOUT=*,DCB=(LRECL=80,BLKSIZE=80,RECFM=F)
```

You can also write FSTRACE, FSTRACE3, and FSTRACE4 results to an MVS sequential data set. First, allocate the FSTRACE data set in a prior batch step (as shown) or in ISPF:

```
//ALLOC    EXEC PGM=IEFBR14
//FSTRACE  DD   DISP=(,CATLG),DSN=userid.FSTRACE,
//              UNIT=SYSDA,VOL=SER=USERM1,SPACE=(TRK,(5,5)),
//              DCB=(LRECL=80,BLKSIZE=80,RECFM=F)
                      .
                      .
                      .
```

Then, allocate the trace data set with DISP=MOD in the batch FOCUS JCL:

```
                      .
                      .
                      .
//FOCBATCH EXEC PGM=FOCUS
//FSTRACE  DD   DISP=(MOD,KEEP,KEEP),DSN=userid.FSTRACE
```

E SQL Codes and Interface Messages

Topics:

- Common SQL Return Codes
- DB2 and SQL/DS Interface Error Messages

Common SQL Return Codes

Some common SQL codes are:

`+100`	No row meets the search conditions specified in a DELETE, UPDATE, or FETCH operation; or table is empty.
`0`	Successful execution.
`-104`	SQL syntax error. Run trace to determine nature of translation error.
`-204`	The table specified by TABLENAME in the FOCUS Access File does not exist.
`-205`	The SQL column name, defined by the ALIAS parameter in either the FOCUS Master or Access File, does not exist within the target TABLENAME.
`-301` `-302`	The USAGE type or length defined for a field in the FOCUS Master does not conform to the corresponding column definition in the SQL table.
`-309` `-407`	There is a null value within a predicate (-309) or within an update statement (-407) where nulls are not allowed.
`-551` `-552`	You tried to perform an action within SQL for which you are not authorized.
`-601`	You tried to create a table, index, or view using a name that already exists.
`-612`	You tried to create a table that contains duplicate SQL column names. Check each Master File segment for unique ALIAS values.
`-710`	The private DBSPACE you tried to access is currently being used by another user (SQL/DS only).
`-803`	You tried to include or update a value within a column for which a UNIQUE INDEX exists, and that value is already present.
`-904`	A resource is unavailable. Refer to the DB2 messages and codes manual in order to evaluate the situation.

`-905`	A resource limit based on the DB2 RLST Governor has been exceeded. There are rows in the SYSIBM.DSNRLST00 table that pertain to this resource and the limits placed on it. Consult your DB2 DBA to find out what action is necessary.
`-911` `-913`	The application was the victim in a deadlock or experienced a timeout.
`-923`	DB2 is not operational.
`-934`	The SQL/DS bootstrap module, ARISRMBT, was not found while loading the FOCUS/SQLDS interface module. Run the SQLINIT EXEC.
`-940`	The SQL/DS virtual machine is not operational.

Note: If the Interface is using the DB2 Call Attachment Facility (CAF), and an attempt to communicate with a DB2 subsystem results in a CAF error (for example, the subsystem specified does not exist), the Interface will return the decimal representation of the CAF error code to FOCUS. The hexadecimal representations of these codes range from 00C10002 (decimal 12648450) to 00C10824 (decimal 12650532) and may be found, accompanied by their explanations, in the IBM *DB2 Messages and Codes* manual. This IBM manual should be consulted for information concerning any of these error codes that are returned by the Interface.

DB2 and SQL/DS Interface Error Messages

If you need to see the text or explanation for any Interface error message, you can display it online in your FOCUS session or find it in a standard FOCUS ERRORS file.

Messages 1351-1800 are stored in FOCSQL ERRORS for CMS and ERRORS.DATA(FOCSQL) for MVS; messages 2501-2621 are stored in FOCCD ERRORS for CMS and ERRORS.DATA(FOCCD) for MVS.

Starting in Version 6.8, all of the Interface error and warning messages were expanded in the same fashion as the core FOCUS errors and warnings. Issuing a ? n query (where n is the message number) at the FOCUS command level displays the message number and text along with a detailed explanation of the message. For example, if you receive the message

`(FOC1369)  STATIC LOAD MODULE AND FOCCOMP OUT OF SYNCH` ***`name`***

you can enter

```
? 1369
```

to get this explanation:

`(FOC1369)  STATIC LOAD MODULE AND FOCCOMP OUT OF SYNCH` ***`name`***

When a FOCEXEC which was compiled using the static SQL option was run, the timestamps for the FOCCOMP and its corresponding static SQL load module did not match. Check current FOCCOMP and STUBLIB allocations, and recompile the FOCEXEC with the static SQL option activated if necessary.

F Additional DB2 Topics

Topics:

- The DB2 Distributed Data Facility
- DRDA Support
- Read-only Access to IMS Data From DB2 MODIFY Procedures

The DB2 Distributed Data Facility

The Interface fully supports the DB2 Distributed Data Facility (DDF). The following sections describe Interface support for DB2 DDF.

File Descriptions

To identify the DB2 subsystem, include the DB2 LOCATION attribute in the TABLENAME parameter of the Access File. This identifies the table to FOCUS as a remote table. The syntax is

```
TABLENAME=[location.][creator.]table
```

where:

location
Is the DB2 subsystem location name; 16 characters maximum.

creator
Is the authorization ID of the table's creator; 8 characters maximum.

table
Is the name of the RDBMS table or view; 18 characters maximum.

Note: The TABLENAME value can have be a maximum of 44 characters (including the required periods).

Tables

The Interface allows you to join tables at different locations. FOCUS performs the join since DB2 does not permit a single SQL statement to reference data at more than one location.

The Interface determines that FOCUS must manage the join based on the presence and value of the LOCATION attribute in the TABLENAME parameter of one or more of the tables referenced in the request. If all remote tables have the same LOCATION attribute, the Interface optimizes the join to the RDBMS. If you use local aliases for remote tables (and, therefore, have no LOCATION attribute in the table name), you must SET OPTIMIZATION=OFF for each request that joins tables from more than one location.

FOCUS automatically appends a "FOR FETCH ONLY" clause to SELECT statements generated by TABLE requests. This clause assists the DB2 optimizer in access path selection and offers substantial performance improvements.

DRDA Support

The DB2 Data Driver supports IBM's Distributed Relational Database Architecture (DRDA), including:

- The Level 1 DRDA CONNECT command.
- Level 2 commands included in DB2 Version 3 and above, and the SET CURRENT PACKAGESET command included in DB2 Version 2 Release 3 and above.

Level 1 DRDA Support: CONNECT

Users can change their default database server with the CONNECT command.

In addition, CONNECT authority controls access to the SQL/DS RDBMS. The administrator (or authorized person) may establish one ID and password for a group of users. Assigning CONNECT authority to the group ID, rather than the individual users, simplifies the task of administering privileges (see Chapter 3, *Security*).

The syntax of the CONNECT command is

```
SQL [target_db] CONNECT [TO dbname|RESET]
```

where:

target_db
: Indicates the target RDBMS. Valid values are DB2 or SQL/DS. Omit if you previously issued the SET SQLENGINE command.

`dbname`

Is the DB2 location identifier or the name of the SQL/DS database to which you wish to connect. If dbname is not specified, for SQL/DS connection is to the default database established by the SQLINIT EXEC.

`RESET`

Applies to DB2 only. Reconnects to the local DB2 server.

You can include the CONNECT command in a FOCEXEC such as the PROFILE FOCEXEC. The FOCEXEC may be encrypted to prevent unauthorized viewing of the SQL/DS authorization ID and password.

Note: For DB2

- The Interface must be installed using the DB2 BIND PACKAGE command; consult with your on-site DBA.
- For more information, consult the IBM DB2 Version 2 Release 3 *SQL Reference* manual.

Level 2 DRDA Support

Level 2 DRDA commands include:

- The SET CONNECTION and RELEASE commands for controlling connection to a remote server.
- The SET CURRENT PACKAGESET command.

The syntax for the SET CONNECTION and RELEASE commands is

```
SQL [DB2] SET CONNECTION server_name
```

```
SQL [DB2] RELEASE {server_name|option}
```

where:

`server_name`

Is the location name of any valid DRDA database server.

`option`

Can be one of the following:

```
CURRENT
ALL [SQL]
ALL PRIVATE
```

Note: Omit the DB2 target database qualifier if you previously issued the SET SQLENGINE command for DB2.

You can use the SET CURRENT PACKAGESET command to switch application packages during a session. The effect is similar to the SET PLAN command, but does not require that the DB2 thread be closed and reopened (for example, when switching Isolation Levels).

The syntax is

```
SQL [DB2] SET CURRENT PACKAGESET {packageset_name|USER}
```

where:

`packageset_name`
: Is the name of any package set previously bound into the current DB2 plan.

`USER`
: Indicates the primary authorization ID.

Use of DRDA and the SET CURRENT PACKAGESET command requires that the Interface be installed using the DB2 BIND PACKAGE command. For more information, consult with the DBA at your site or refer to the IBM *DB2 Application Programming and SQL Guide*.

For a complete description of these commands, refer to the IBM *DB2 SQL Reference* for DB2 Version 3 and above.

Read-only Access to IMS Data From DB2 MODIFY Procedures

The DB2 Interface not only allows FOCUS MODIFY procedures to view and/or update data residing in DB2 tables, it also gives them read-only access to IMS files. You can issue FIND and LOOKUP commands against IMS files and use MATCH and NEXT commands to display IMS data. FOCUS can access data from both DB2 and IMS transparently, postponing or eliminating the need to physically move IMS files to DB2.

Prerequisites

Read-only access to IMS data from DB2 MODIFY procedures requires installation of the FOCUS IMS/DB Interface in addition to the FOCUS DB2 Interface. Both products are available from Information Builders. Please see the *FOCUS for IBM Mainframe DB2 and SQL/DS Read/Write Interface Installation Guide* and the *FOCUS for IBM Mainframe IMS/DB Interface User's Manual and Installation Guide* for installation instructions.

Note:

- This feature requires that the DB2 Interface be installed to use the Call Attachment Facility (CAF). You can verify the attachment facility by issuing the SQL DB2 ? command; Call Attach should be ON. At the present time, the Interface does not support the use of the IMS Attachment Facility for this feature.

- To issue MODIFY write commands (INCLUDE, UPDATE, DELETE) against DB2 tables, you must install the DB2 Read/Write Interface. If your MODIFY procedure only displays data (MATCH, NEXT, FIND, LOOKUP), the DB2 Read-only Interface is sufficient.

Implementation

MODIFY commands accessing IMS data behave similarly to MODIFY commands accessing the FOCUS DBMS. If you are not familiar with FOCUS MODIFY commands, consult Chapter 12, *Maintaining Tables With FOCUS* and the *FOCUS for IBM Mainframe User's Manual*.

If an IMS file participates in a COMBINE structure with a DB2 table, the MODIFY MATCH, NEXT, REPOSITION, and FIND commands can access the IMS data. If you dynamically JOIN the IMS file to a DB2 Master File, you can LOOKUP data values in the IMS file. The following sections discuss these two techniques.

MODIFY Subcommands With a COMBINE Structure

If you COMBINE the IMS file with a DB2 table, several MODIFY subcommands can access the IMS data. The syntax is

```
COMBINE FILES  file1 [PREFIX pref1|TAG tag1] [AND]
   .
   .
   .
               filen [PREFIX prefn|TAG tagn] AS asname
```

where:

file1 - filen

Are the Master File names of the tables you want to modify. You can specify up to 16 Master Files.

pref1 - prefn

Are prefix strings for each file; up to four characters. They provide uniqueness for fieldnames. You cannot mix TAG and PREFIX in a COMBINE structure. Refer to the *FOCUS for IBM Mainframe User's Manual* for additional information.

tag1 - tagn

Are aliases for the table names; up to eight characters. FOCUS uses the tag name as the table name qualifier for fields that refer to that table in the combined structure. You cannot mix TAG and PREFIX in a COMBINE, and you can only use TAG if FIELDNAME is set to NEW or NOTRUNC.

AND

Is an optional word to enhance readability.

`asname`

Is the required name of the combined structure to use in MODIFY procedures and CHECK FILE commands.

Note:

- Any attempt to perform an INCLUDE, UPDATE, or DELETE operation on an IMS segment results in an error message.
- Avoid using CRTFORM * on IMS segments.

Once you issue the COMBINE FILE command, you can access the IMS files in the structure with the following MODIFY commands:

`MATCH`

The MATCH statement selects specific segment instances based on their values. It compares field values in the instances with incoming data values.

The Interface passes a MATCH on a full key that is defined as .IMS, .HKY, or .KEY, directly to the IMS DBMS. This is the most efficient access method; the IMS/DB Interface issues a GET UNIQUE.

A MATCH on a non-key or partial key field repositions to the beginning of the chain and searches forward for the specified value. This search is not passed to the IMS DBMS and, therefore, is less efficient. The Interface issues GET NEXT commands until the match condition is met or no records remain.

When you specify a MATCH with both a key and a non-key field, the MATCH condition for the key is passed to the IMS DBMS for retrieval; then the Interface does the sequential forward search for the non-key value. The Interface issues a GET UNIQUE for the qualified key, applies the condition for the non-key value based on the qualified key, and then issues GET NEXT commands for the qualified value.

`NEXT`

The NEXT statement selects the next segment instance after the current position, making the selected instance the new current position. The current position depends on the execution of the MATCH and NEXT statements:

- If a MATCH or NEXT statement selects a segment instance, that instance becomes the current position within the segment.
- If a MATCH or NEXT statement selects a parent instance of a segment chain, the current position is before the first instance in that chain.
- At the beginning of a request, the current position is in the root segment before the first instance.

NEXT processing with a MATCH statement is identical to NEXT processing without a MATCH statement. If a current position has not been established, the Interface issues a GET UNIQUE to obtain the first record and then issues GET NEXT calls to obtain remaining records.

NEXT processing after MATCH on a non-key or partial key produces the same results. It is important to realize that the MATCH on the non-key is not passed to the IMS DBMS. The Interface repositions from the beginning of the chain and searches forward until the condition is met. Therefore, for maximum efficiency, consider matching on the primary key.

`REPOSITION`	The REPOSITION command sets the current position to the beginning of the IMS segment chain you are traversing or to the beginning of the chain of any of the parent instances along the segment path. Refer to the *FOCUS for IBM Mainframe User's Manual* for additional information.
`FIND`	The FIND function tests for the existence of indexed values in IMS files. COMBINE must be in effect in order to use FIND. Refer to the *FOCUS for IBM Mainframe User's Manual* for additional information.

You can also use the following MODIFY commands for DB2 tables in the COMBINE structure (refer to Chapter 12, *Maintaining Tables With FOCUS*, for additional information):

- MATCH, NEXT, REPOSITION, INCLUDE, DELETE.
- FIND. You cannot use LOOKUP, which requires a dynamic JOIN, in the same MODIFY because you cannot issue a JOIN when a COMBINE is in effect.

The LOOKUP Function With a Dynamic JOIN

The LOOKUP function retrieves data values from files joined dynamically by the FOCUS JOIN command (see Chapter 8, *Advanced Reporting Techniques*). When you join a DB2 Master File to an IMS file, you can LOOKUP either DB2 or IMS data.

You cannot issue LOOKUP if the MODIFY contains commands that require a COMBINE (for example, FIND of IMS or DB2 data), or if it contains MATCH and/or NEXT commands against IMS data.

LOOKUP for IMS files supports the extended syntax parameters GE and LE, while LOOKUP for DB2 data does not. Refer to the *FOCUS for IBM Mainframe User's Manual* for additional information. Also consult Chapters 8, *Advanced Reporting Techniques*, and 12, *Maintaining Tables With FOCUS*, of this manual, and the *FOCUS for IBM Mainframe IMS/DB Interface User's Manual and Installation Guide*.

Run-time Requirements

After the IMS/DB and DB2 Interfaces are installed, you must create a CLIST or JCL to invoke this feature. Subsequent sections outline JCL and CLIST preparation.

JCL Preparation for Batch Environments

The following JCL runs a batch FOCUS job that uses both the IMS/DB and DB2 Interfaces to provide read-only access to IMS files during update of DB2 tables. You must concatenate the FOCLIB.LOAD library with DDNAME STEPLIB since the IMS software program, DFSRRC00, searches STEPLIB only for libraries that are called. You can concatenate the FOCUS Interface module, IMS, with DDNAME STEPLIB or USERLIB.

This JCL is only a model. Before executing it, you must create an appropriate job card and modify the JCL to conform to your site's specifications

```
//JOB card goes here
//BATIMS   EXEC PGM=DFSRRC00,PARM='DLI,FOCUS,PSBNAME'
//STEPLIB  DD   DISP=SHR,DSN=prefix.FOCLIB.LOAD
                DISP=SHR,DSN=DSN510.SDSNLOAD
//USERLIB  DD   DISP=SHR,DSN=prefix.IMS.LOAD
           DD   DISP=SHR,DSN=prefix.FOCSQL.LOAD
           DD   DISP=SHR,DSN=prefix.FOCLIB.LOAD
           DD   DISP=SHR,DSN=prefix.FUSELIB.LOAD
//DFSRESLB DD   DISP=SHR,DSN=IMSVS.RESLIB
//ERRORS   DD   DISP=SHR,DSN=prefix.ERRORS.DATA
                DISP=SHR,DSN=prefix.IMS.DATA
//IMS      DD   DISP=SHR,DSN=user.DBDLIB
           DD   DISP=SHR,DSN=user.PSBLIB
//FOCPSB   DD   DISP=SHR,DSN=user.FOCPSB(PSBNAME)
//MASTER   DD   DISP=SHR,DSN=user.MASTER.DATA
//FOCSQL   DD   DISP=SHR,DSN=user.FOCSQL.DATA
//FOCEXEC  DD   DISP=SHR,DSN=user.FOCEXEC.DATA
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   *
           FOCUS request goes here.  For instance,
TABLE FILE imsfile
PRINT field1
END
FIN
```

where:

prefix

Is the high-level qualifier for your FOCUS production libraries.

user

Is the qualifier for a private version of a library.

Note: For a description of other IMS environments and their corresponding JCL requirements, see the *FOCUS for IBM Mainframe IMS/DB Interface User’s Manual and Installation Guide*.

CLIST Preparation for Interactive Environments

You can use this feature interactively if you call it from a CLIST or REXX EXEC. You can allocate the FOCUS load libraries directly in your CLIST or REXX EXEC. This CLIST is only a model; edit it to conform to your site's standards

```
PROC 0
CONTROL MSG NOLIST NOFLUSH
ALLOC F(STEPLIB)      DA('prefix.FOCLIB.LOAD')    SHR REUSE
ALLOC F(USERLIB)      DA('prefix.IMS.LOAD'        -
                         'prefix.FOCSQL.LOAD'     -
                         'prefix.FOCLIB.LOAD'     -
                         'prefix.FUSELIB.LOAD')   SHR REUSE
ALLOC F(DFSRESLB)     DA('IMSVS.RESLIB')          SHR REUSE
ALLOC F(FOCEXEC)      DA('user.FOCEXEC.DATA')     SHR REUSE
ALLOC F(MASTER)       DA('user.MASTER.DATA')      SHR REUSE
ALLOC F(FOCSQL)       DA('user.FOCSQL.DATA')      SHR REUSE
ALLOC F(ERRORS)       DA('prefix.ERRORS.DATA'     -
                         'prefix.IMS.DATA')       SHR REUSE
ALLOC F(FOCPSB)       DA('user.FOCPSB(PSBNAME)')  SHR REUSE
ALLOC F(IMS)          DA('user.PSBLIB'            -
                         'user.DBDLIB')           SHR REUSE
CALL  'IMSVS.RESLIB(DFSRRC00)'  'DLI,FOCUS,PSB'
```

where:

prefix

Is the high-level qualifier for your FOCUS production libraries.

user

Is the high-level qualifier for a private version of a library.

Note: For a description of other IMS environments and their corresponding CLIST requirements, see the *FOCUS for IBM Mainframe IMS/DB Interface User's Manual and Installation Guide*.

Index

Reader Comments

In an ongoing effort to produce effective documentation, the Documentation Services staff at Information Builders welcomes any opinion you can offer regarding this manual.

Please use this form to relay suggestions for improving this publication or to alert us to corrections. Identify specific pages where applicable. Send comments to:

Corporate Publications
Attn: Manager of Documentation Services
Information Builders
Two Penn Plaza
New York, NY 10121-2898

or FAX this page to (212) 967-0460, or call **Sara Elam** at (212) 736-4433, x**3207**.

Name: ____

Company: ____

Address: ____

Telephone: ______________________________ Date: ____________________

Comments:

Reader Comments